DOCUMENT RESUME

ED 090 972 IR 000 583

AUTHOR	Lerner, Edward Mark
TITLE	On-Line Filing (OLF); A Program Package for Student Records.
INSTITUTION	Illinois Univ., Urbana. Dept. of Computer Science.
REPORT NO	UIUCDCS-R-73-582
PUB DATE	Oct 73
NOTE	101p.; Master's Thesis, University of Illinois
EDRS PRICE	MF-$0.75 HC-$5.40 PLUS POSTAGE
DESCRIPTORS	*Academic Records; Computer Programs; Educational Counseling; Grade Point Average; Grades (Scholastic); Information Processing; *Information Retrieval; Information Storage; *On Line Systems; Program Descriptions; *Student Personnel Services; Student Records
IDENTIFIERS	*Academic Advising Aids; OLF; On Line Filing

ABSTRACT

A prototype package of interactive computer programs for automating the information processing needs of the academic advisor in large educational systems is described in this report. This on-line filing (OLF) package is basically a program which retrieves an existing student record and allows easy interaction with a set of subroutines that perform a variety of operations on the record. The OLF package is written in CALL/360-OS FORTRAN for use on the University of Illinois IBM 360/75 facility. Operations that can be performed with the subroutines described include the calculation of grade point averages and other simple statistics. The system is structured in such a way that additional routines can be easily incorporated into the existing package. There are four appendixes that are of high importance: A) computing services office documents relevant to OLF; B) OLF operations manual; C) new Processors; D) OLF program listings. (WDR)

ED 090972

Report No. UIUCDCS-R-73-582

ØN-LINE FILING (ØLF)
A PRØGRAM PACKAGE FØR STUDENT RECØRDS

by

Edward Mark Lerner

October 1973

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

IR 000583

ED 090972

Report No. UIUCDCS-R-73-582

ØN-LINE FILING (ØLF)
A PRØGRAM PACKAGE FØR STUDENT RECØRDS*

by

Edward Mark Lerner

U S DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

October 1973

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by the Department of Computer Science and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, October 1973.

ACKNOWLEDGMENT

The author would like to take this opportunity to thank the many people who contributed to the success of this project: his advisor, Dr. H. G. Friedman, Jr., for his guidance; the systems programmers of the Computing Services Office for their assistance and cooperation; Mrs. Vivian Alsip for typing this report.

The author especially wishes to thank his wife for her patience and encouragement, without which this project might not have been completed.

TABLE OF CONTENTS

		Page
1.	INTRODUCTION	1
2.	SYSTEM CONSIDERATIONS	5
3.	SEARCH STRATEGY & FILE STRUCTURE	9
	3.1 Basic Search Methods	9
	3.2 The ØLF Tree	9
	3.3 File Names	15
	3.3.1 Student Files	15
	3.3.2 Directory Files	16
	3.3.3 Other Files	17
	3.4 Search Strategies	17
4.	PROCESSORS	22
	4.1 The Processor Interface	22
	4.2 Sample Processors	25
	4.2.1 CREATE FILE (CRFILE)	25
	4.2.2 DELETE FILE (DLFILE)	26
	4.2.3 GPA CALCULATOR (CALC)	27
	4.2.4 A SAMPLE STATISTICS PROCESSOR (USERA)	27
5.	SAFETY & SECURITY	29
6.	UTILITIES PROGRAMS	32
7.	CONCLUSION	33
APPENDIX		
A.	COMPUTING SERVICES OFFICE DOCUMENTS RELEVANT TO ØLF	34
B.	ØLF OPERATIONS MANUAL	39
	B.1 Introduction	39
	B.2 Search Commands	39
	B.3 Data Statements	41
	B.4 Errors	42
	B.5 Input Modes	44
	B.6 Use of ØLF	45
	B.7 Present Processors	48

Page

C. NEW PROCESSORS. 53

C.1 Changes in ØLF 53
C.2 Scratch Files. 54
C.3 Programming Aids 54
C.4 Information. 54

D. ØLF PROGRAM LISTINGS. 56

BIBLIOGRAPHY 94

LIST OF FIGURES

		Page
1.	Graduation Requirements Form--Department of Computer Science (Side 1).	2
2.	Graduation Requirements Form--Department of Computer Science (Side 2).	3
3.	The PLØRTS Tree	6
4.	The ØLF Tree.	10
5.	ØLF within the PLØRTS Tree.	11
6.	ØLF Directories	13
7.	The Student File.	14
8.	Interconnections in the ØLF Tree.	18
9.	Tree Pointers	21
10.	Processor Access Rights Table (PART).	23
11.	Search Commands	40
12.	Data Statements	42
13.	ØLF Statement Hierarchy	44
14.	Sample CALC Data Statements	51
15.	Sample Use of USERA	52

1. INTRODUCTION

At present, the various departments of the University of Illinois (UI) manually maintain student records. Courses--scheduled, in progress, and completed--and grades received are typically recorded on a form like the one shown in Figures 1 and 2, used by the Department of Computer Science. Faculty members (advisors) who help plan students' programs use the information on these forms to arrive at their recommendations for future coursework.

The advisor's duties are complicated by several considerations. Often he must counsel students in different curricula; occasionally, as in the Department of Computer Science, these curricula reside in different colleges. A student is allowed to meet either current requirements or the requirements which were in effect when he entered his curriculum--whichever are least stringent. The amount of information with which an advisor must be familiar is clearly large.

To lighten the burden on the advisor, it would be advantageous to automate this information retrieval and data processing system. A prototype on-line filing (ØLF) program package has been written to provide the framework for such a system.

Viewed in its simplest form, ØLF is a program which retrieves an existing student record. It is designed to allow secure and easy interaction with subroutines--data processors--requiring different access rights (to both data files and to other processors) and search strategies (all students, students of one department, etc.). Some of the functions necessary for a complete system include file insertion/deletion, program approval (prerequisite checking), and grade point average (GPA) calculation. By incorporating

Name: Social Sec. No.:

High School: Rank: Class: Percentile:

Transfer from: Hours: G.P.A.:

Entered U of I: Entered Curriculum:

Candidate for Distinction ☐

	Hours Cr. Grade	Hours Cr. Complete
A. MAJOR		
1. Mathematics	(31)	☐☐
120		
130/131		
140/141		
341 (345/349)		
342		
317 (319)		
318		
347		
348/361 (363)		
2. Computer Science	(13)	☐☐
121		
201		
287		
293/294/301/310		
B. MINOR	(8)	☐☐
C. ELECTIVES	(24 approx)	☐☐

	Hours Cr. Grade	Hours Cr. Complete
D. COLLEGE REQUIREMENTS		
1. Physical Ed.	(4)	☐☐
2. Rhetoric	(6)	☐☐
3. Foreign Language	(16)	☐☐
101		
102		
103		
104		
4. Biological Science	(8)	☐☐
5. Social Science	(8)	☐☐
6. Humanities	(6)	☐☐

Graduated: Hours: G.P.A.:

Figure 1. Graduation Requirements Form--
Department of Computer Science (Side 1)

MATHEMATICS AND COMPUTER SCIENCE CURRICULUM

Notes on Graduation Requirements Form

Transfers:
Enter, in the appropriate section, the number of hours credit together with a T for grade for all courses transferred onto the student's record.

A: The courses marked on the form are requirements, where a / indicates an alternative and courses in parentheses, if taken before entry to the curriculum, are alternatives to the immediately preceding course on the left.

D1: Four semesters of P.E. required, no credit hours.

D3: Entrance to the four course sequence in a single foreign language may be made at a point determined by a proficiency examination, and the first course taken may be for 0 credit.

D4, D5, D6: Refer to LAS Student Handbook, 1969-70, pp. 8-9.

Advanced Courses:
The requirement of 30 hours in advanced courses is automatically satisfied by the addition of 1 more 200 or 300 level course.

Completed Requirements:
As requirements for each section are completed, check (✓) the appropriate "completion" square and enter the number of hours of credit gained in that section. Indications of the minimum credit requirements are shown in parentheses.

Comments:
Requirement D1 is no longer in effect.

Figure 2. Graduation Requirements Form--
Department of Computer Science (Side 2)

write-protection, search control, and I/O management within ØLF, it becomes safe to allow users to write processors. ØLF, therefore, makes possible the compilation of many presently unavailable statistics.

The ØLF system and a GPA calculator, file insertion/deletion processors, and a sample statistics processor designed to use ØLF are discussed below.

2. SYSTEM CONSIDERATIONS

PLØRTS, the filing, remote job entry, and timesharing facility of the UI uses an IBM 360/75 interfaced to teletype (TTY) and IBM 2741 terminals through a PDP-7. PLØRTS files are maintained on magnetic discs. Interactive jobs are limited to 100K bytes of main memory. If several users try concurrently to use a single file, a copy is loaded into core storage for each. The PLØRTS monitor prevents one user from overwriting a file being read by another user.

At present, BASIC and FØRTRAN are supported in their CALL/360-ØS implementations (with a few minor changes to follow the conventions of PLØRTS). ØLF has been programmed in FØRTRAN because of the inferior character-manipulation features of CALL/360-ØS BASIC.

Access to PLØRTS requires knowledge of two items: a valid PLØRTS problem specification number (PS#) used for accounting purposes and a name recognized as that of a user under that PS#. PLØRTS files form a tree as shown in Figure 3. A complete name has the form

PS#.username.level1name.level2name. ... .levellastname

where each level name is a character string of eight or fewer EBCDIC characters (exclusive of blank, comma, or period). File names created within a program can contain "characters" which are not available on a terminal; any such name can then be accessed only by a program. In order that a file be accessible by program, the part of its name corresponding to the underlined portion of the general form above cannot exceed eight characters. The more qualified (the more levels to) a file's name, the greater the access time for the file.

A PLØRTS user is given full access (read/write/execute privileges)

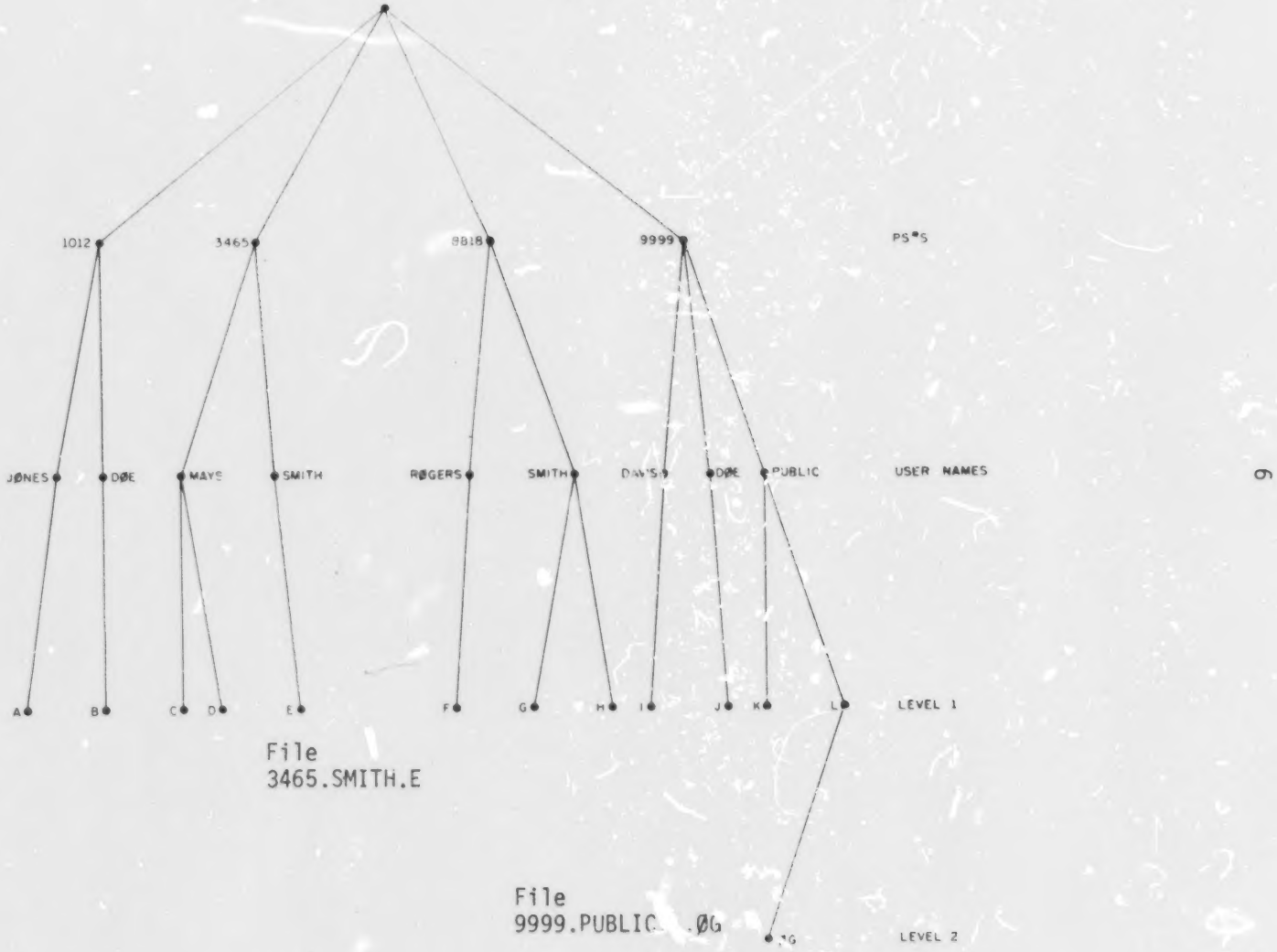


Figure 3. The PLØRTS Tree

only to the branch of the tree named by the PS# and name with which he logged in. (To access a file farther down on that branch, he need specify only the remaining part of its name.) He can read (copy) another user's file if he knows the complete file name. This is the basic PLØRTS security arrangement.

Execution of TSBATCH. a catalogued procedure in the batch processing system, makes card image input appear to the PLØRTS monitor to be a PLØRTS terminal. This has three advantages. First, large amounts of input can be fed to an "interactive" program through the card reader (any mispunched information--hopefully a small subset of the original data--can then be resubmitted in true interactive mode). Second, a PLØRTS program can be run as one job step of a longer batch job. Third, PLØRTS can be used even when the PDP-7 is not available.

Each PLØRTS file is composed of blocks. A block contains, besides information for the monitor, eight or more card images (varying because trailing blanks are suppressed). Files can be created either by the terminal command ØPEN, or, during program execution, by invoking for output the subroutine ØPEN. One block is allocated to a file when it is created; additional blocks are automatically added as needed.

Unfortunately, files can be deallocated only by the terminal command destroy (DEST). However, a program can lose all information in a file without deallocating its disc space by opening the file for output. (If the file has multiple blocks, this maneuver will reduce to one the number of blocks allocated.) An executing job can, however, write onto disc the card image of a DEST command for each obsolete file. A TSBATCH job can then use these records to perform the file deallocation.

As mentioned, PLØRTS files are kept on magnetic discs. Programs are swapped between main memory and disc during execution. Data files appear

sequential to executing programs. In this environment, program structure is optimized with respect to response time by:

1) minimizing the number of file accesses;
2) manipulating files in core storage whenever possible (so that access of file items is direct);
3) compacting the program to minimize the time spent waiting for main memory to become available for reloading.

The methods used in ØLF to implement these goals are discussed below.

3. SEARCH STRATEGY & FILE STRUCTURE

3.1 Basic Search Methods

A program can locate, or determine the absence of, a student's file by generating the appropriate file name and invoking for input the ØPEN subroutine. If, upon return, the error flag is set, the file is absent; otherwise the file has been successfully accessed.

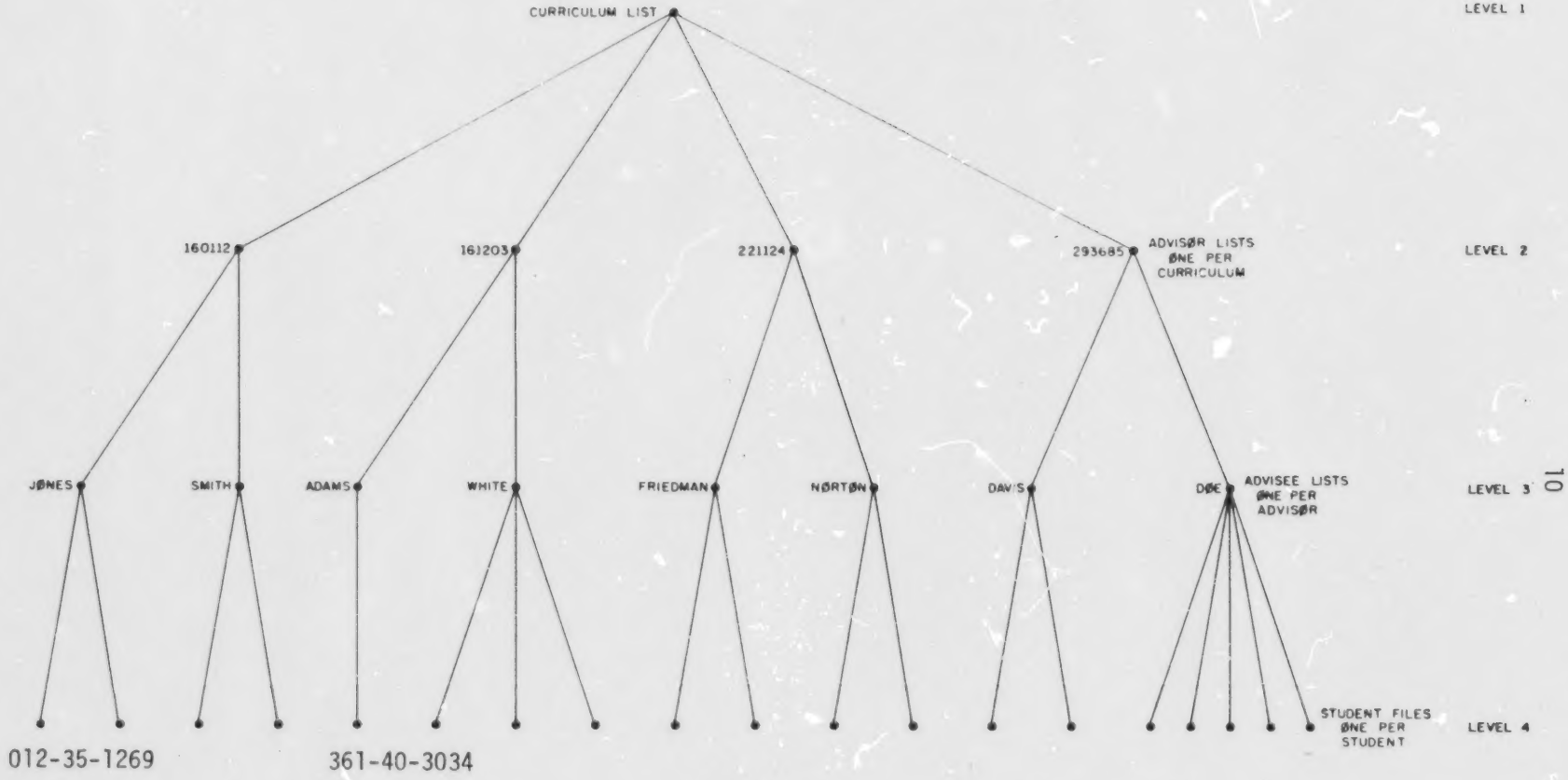
The single search strategy possible using this method requires naming each file individually. An attempt to compile university-wide statistics would require approximately 30,000 such specifications. A prepared list of students would make group access feasible, but preparation of a student list for each group (e.g., each college and curriculum) likely to be accessed collectively would entail an inordinate amount of effort and replication.

A tree structure (e.g., Figure 3) allows use of a family of search strategies. Within a tree, a member is uniquely defined by the concatenation of the name of the leaf with the names of all nodes leading to the leaf. More pertinent to this discussion, a set of members can often be selected by naming one node common to the set--the root of a subtree. A tree designed with thought towards probable access classes will most often be searched by tracing only selected subtrees.

3.2 The ØLF Tree

ØLF student files are accessed through the tree shown in Figure 4. A node is implemented as a directory (list) of its immediate branches. Both student files and directory files of all levels are found in the top level of the PLØRTS tree, as shown in Figure 5. Each ØLF file exists independently on disc.

Each directory file consists of an INTEGER*4 length field and a



This figure does not reflect ØLF file name conventions.

The student with SSN 361-40-3034 has advisor Adams of curriculum 161203.

Figure 4. The ØLF Tree

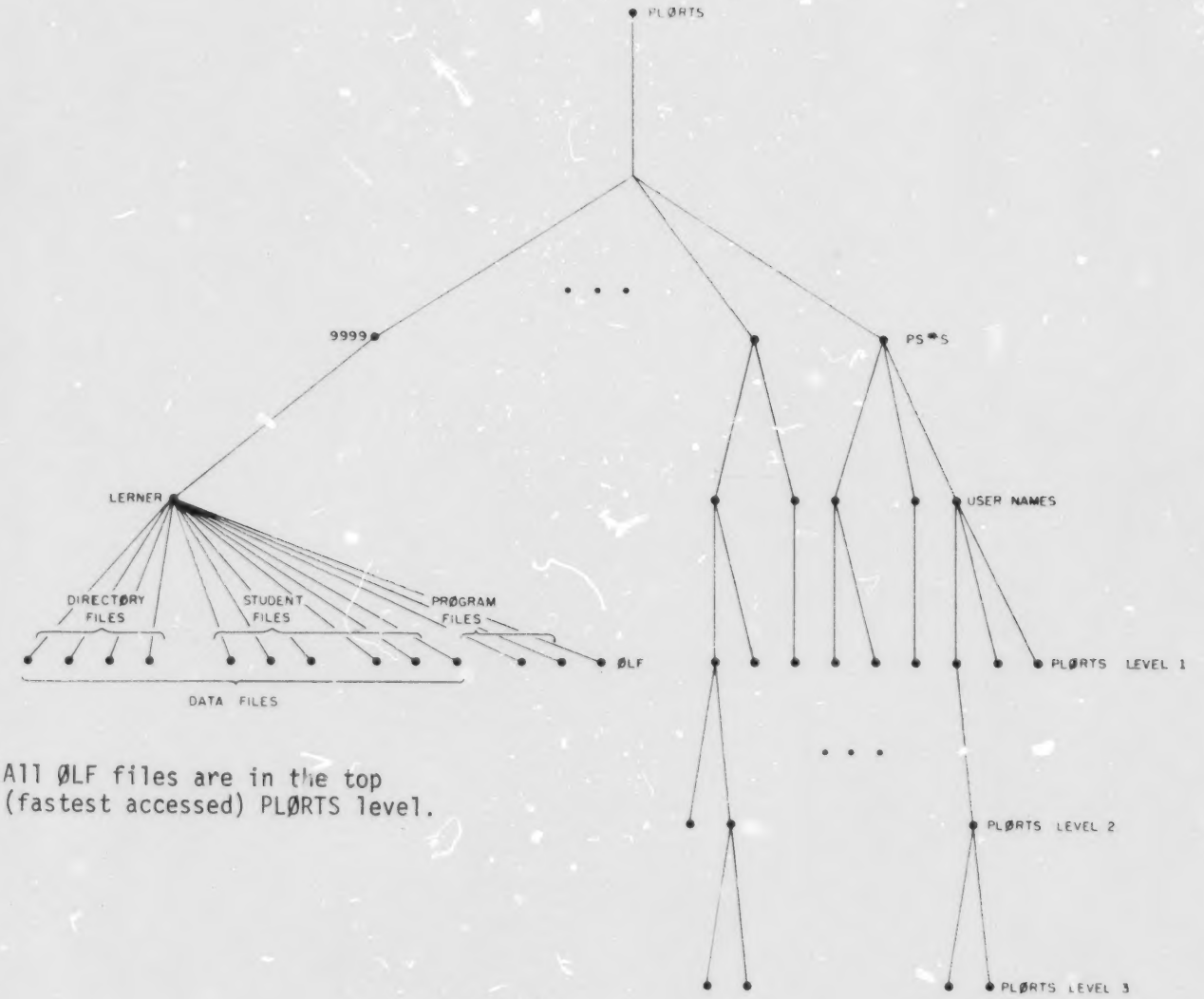


Figure 5. ØLF within the PLØRTS Tree

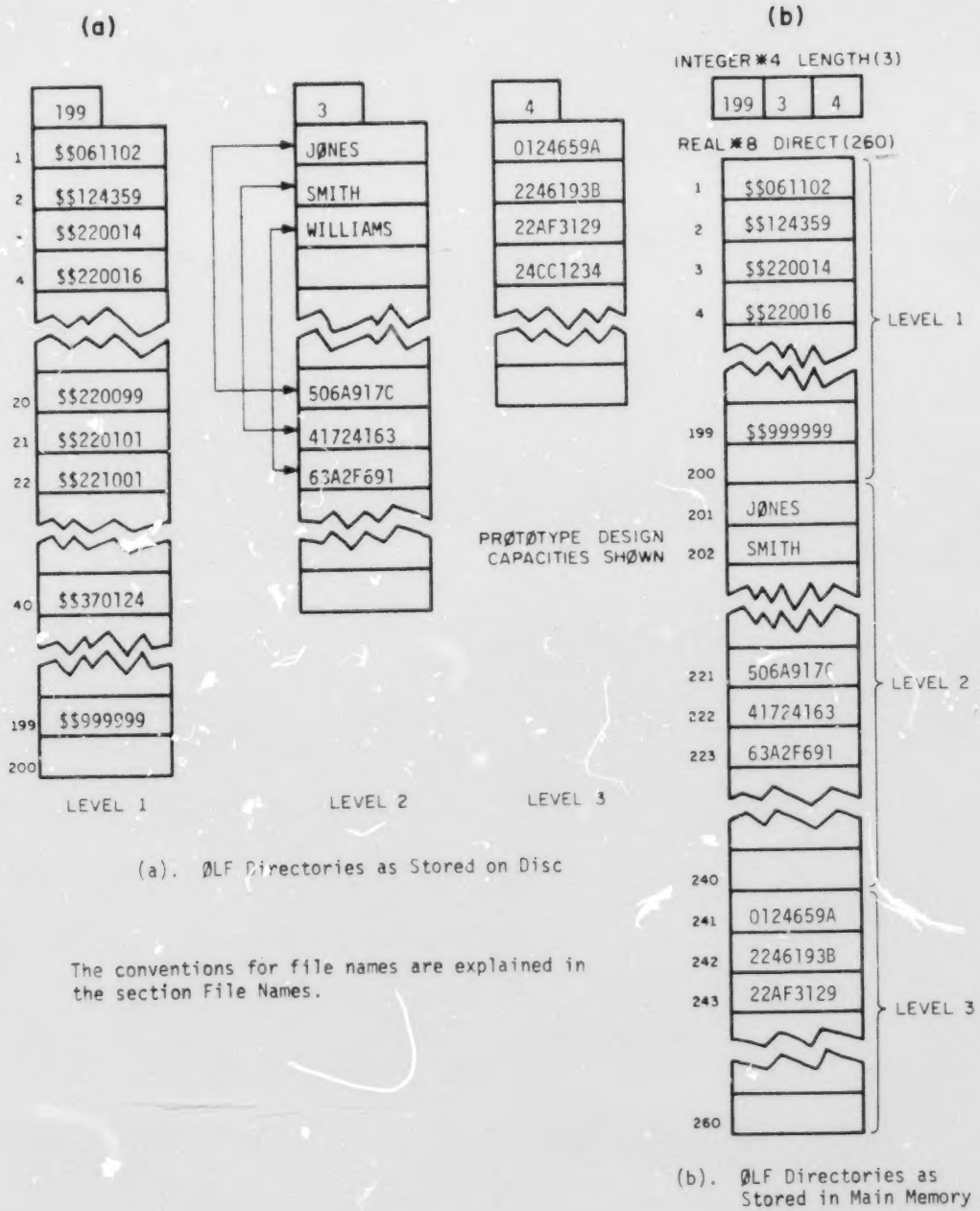
REAL*8 vector. The meaning of each component depends upon the directory level. Figure 6 further illustrates the directory structure.

The single level-one directory lists all curricula whose students have ØLF files. Each element of the vector holds the name of a level-two directory file, formed from a numeric curriculum code. The file names are kept in ascending order. The value of the length field equals the number of curricula listed.

Each level-two directory stores the complete list of undergraduate advisors for the curriculum after which the file is named. Elements in the first half of the vector contain the names, truncated to eight letters, of the advisors in alphabetic order. Each advisor's name has as an associated entry in the second half of the file, the name (obtained from the advisor's social security number) of a level-three directory file. These associated elements have as a group the same relative positions in the bottom half of the vector as do the alphabetized names in the top half. The length field has as its value the number of advisor names (which equals the number of level-three file names) listed.

Each level-three directory holds the list of all students (advisees) of the advisor after whom the file is named. Each vector element stores the name of one student's file, formed from his social security number. The file names are kept in ascending order. The value of the length field indicates the total number of advisees of an advisor, even if the advisor counsels for several curricula (see Search Strategies, below).

The major components of the student file are the header (identification), the general information block (GPA, hours towards degree, etc.), the requirements check-off list, and the course history block. A more detailed description accompanies Figure 7.



(a). ØLF Directories as Stored on Disc

The conventions for file names are explained in the section File Names.

(b). ØLF Directories as Stored in Main Memory

Figure 6. ØLF Directories

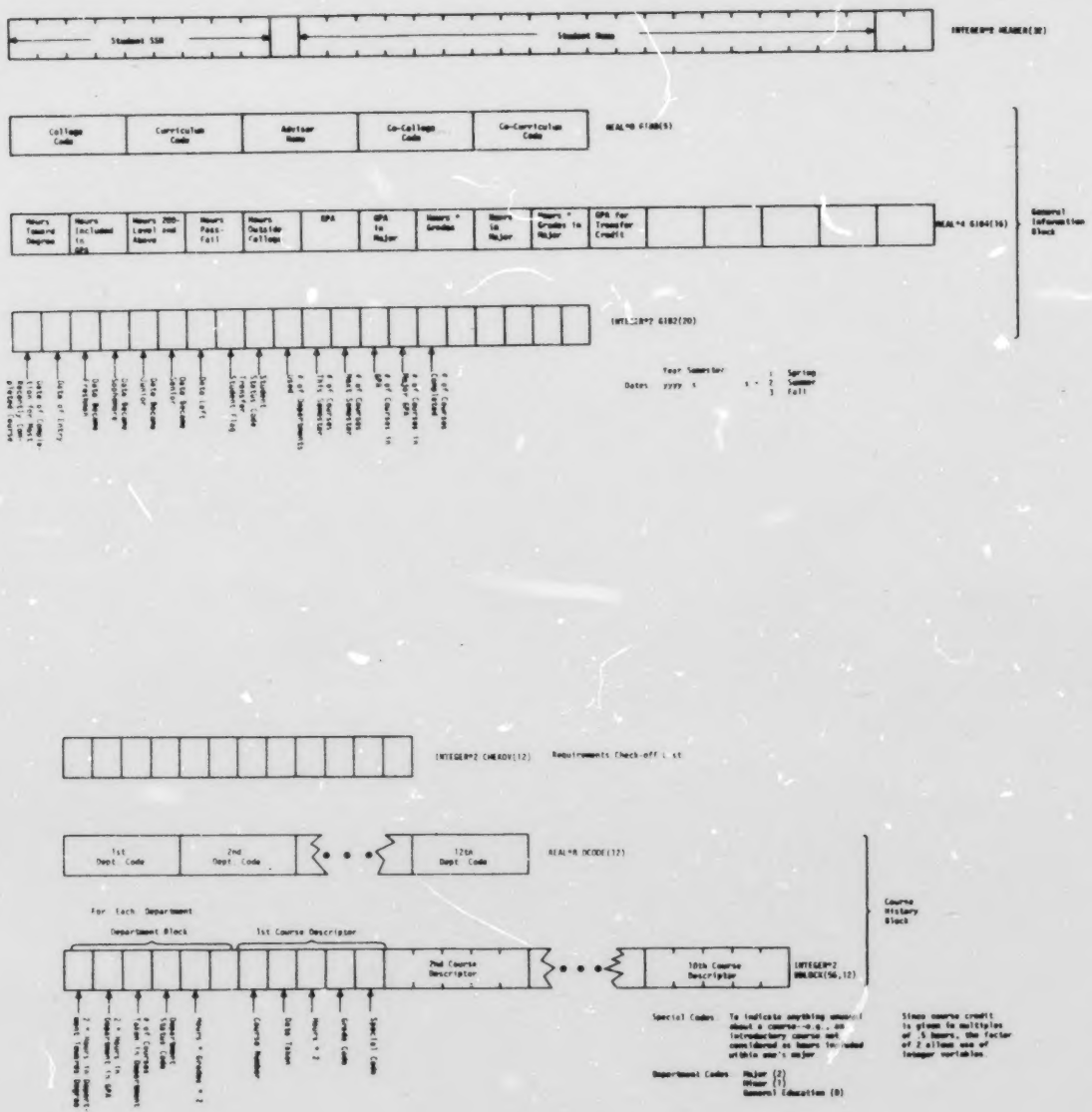


Figure 7. The Student File

The design of the student file is complicated by the goal of allowing for user-supplied processors with unknown data requirements. A small amount of unassigned space is included in the file to be defined as necessary. It is expected that the information which is currently found in the file will be adequate for most purposes.

3.3 File Names

A complete advisory and record keeping system requires student data files, directories, course description tables, and requirements tables for curricula and colleges. A consistent naming convention is needed to avoid duplication of file names. A convenient feature (but see section on SAFETY & SECURITY, below) is direct accessibility from the terminal.

3.3.1 Student Files

UI identifies its students by social security number (SSN). The eight character limit for program-accessible PLØRTS file names precludes direct use of nine digit SSNs as identifiers. The following algorithm is used:

> The SSN, interpreted as a nine digit decimal number, is converted to an eight digit hexadecimal number. Each hexadecimal digit is stored in character representation in a separate byte of a REAL*8 variable. This character string consists entirely of characters available on the terminal and contains no characters that are invalid (or undesirable, like period) in PLØRTS file names.

Student files are on level 4 of the ØLF tree.

3.3.2 Directory Files

The single level-one directory file is the master curriculum list named CURRIC.

The name of each level-two file is derived from a six-digit college/curriculum code. The two leftmost digits indicate the college and the remaining four digits identify the curriculum within the college; curriculum codes shorter than four digits must be extended to four digits by adding non-significant zeroes. A basis for all of a curriculum's file names is formed by storing these digits in character form left-justified in a REAL*8 variable. (No decimal to hexadecimal conversion is required since the code is only six digits long.) The directory name for the curriculum is generated by right-justifying these six digits in another REAL*8 variable and inserting $$ into the two vacant high-order bytes. Other files for this curriculum, if needed, can be named by using other (non-alphanumeric) characters in place of $$. A level-two file named $$220014 contains the advisor list of curriculum 14 of college 22.

The name of each level-three file is generated from an advisor's SSN. It is convenient (see PROCESSORS) that level-three file names be distinguishable from level-four file names, also generated from SSNs.

Nine digit SSNs are bounded by

$$0 \leq SSN \leq 10^9-1$$

while the range for a positive FØRTRAN fullword integer K is

$$0 \leq K < 2^{31}-1 .$$

Adding 10^9 to an advisor's SSN yields a ten digit "SSN" which is convertible to a file name by the algorithm used for level-four file names, but which is easily identified as being a level-three file name.

3.3.3 Other Files

A complete system requires various curriculum, department, and college tables. A basis for curriculum file names is discussed above. Differerent curriculum files resemble level-two directory file names with $$ replaced by other characters.

College files can be similarly formed: **22 might be the name of the requirements file of college 22.

Department files can be formed from the standard UI department name abbreviations. Since these consist of five or fewer letters, names for several departmental files can be generated by addition of special characters.

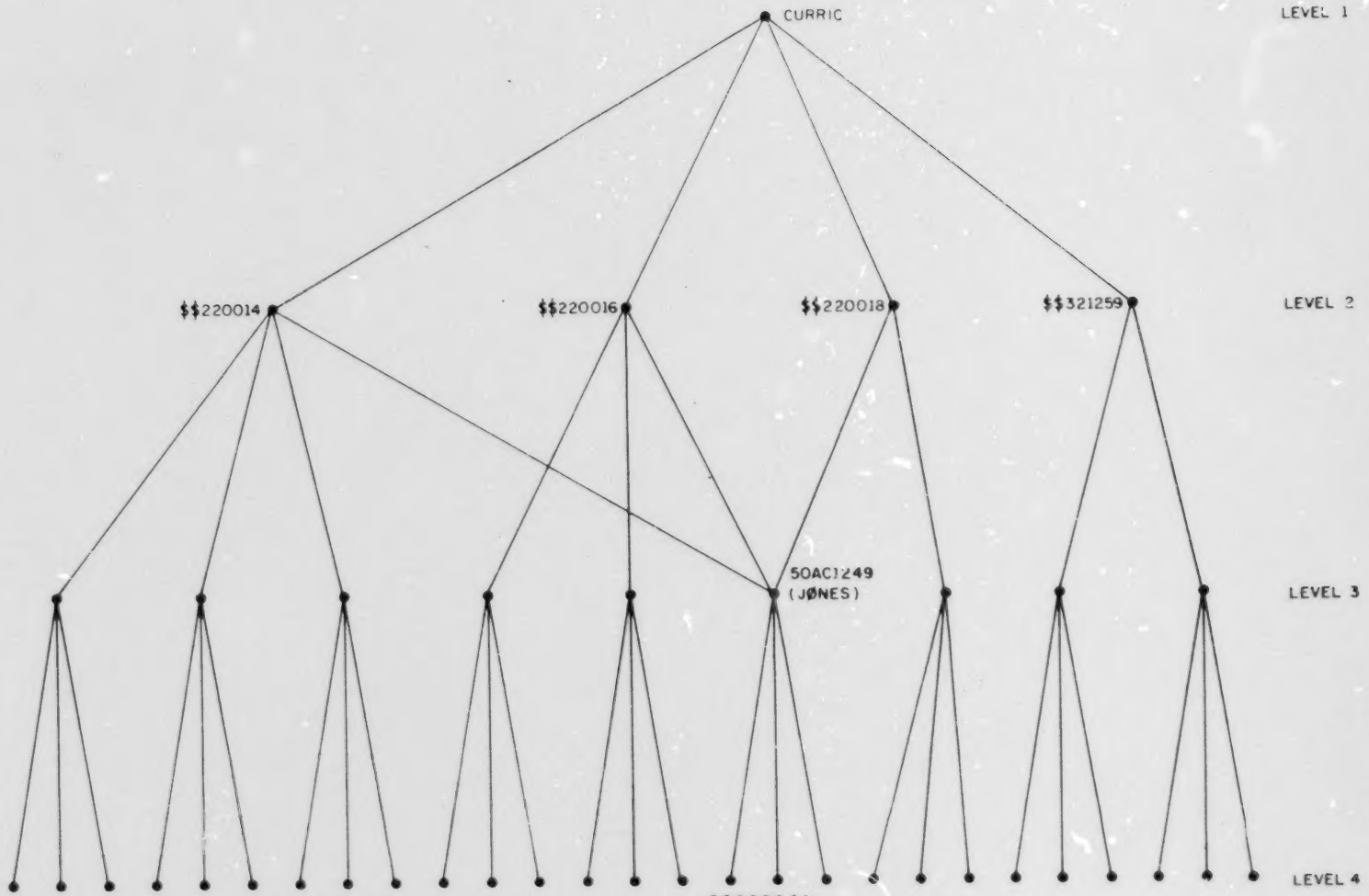
A few files like CURRIC (all are listed in the appendices) have predetermined alphabetic names.

3.4 Search Strategies

Every record search requires a set of search limits. (The implicit limit in searching an unstructured file collection is the collection size, although the search, if successful, may not reach this limit.) The natural search limit in a tree-organized record set is the last common node--the root of the smallest subtree which includes all members of the search class.

The four natural classes in the ØLF tree are: all students, all students within a single curriculum, all students of a common advisor, and an individual student. They correspond, respectively, to last common nodes at levels one, two, three, and four. ØLF uses all four access classes.

A small amount of interconnection (not shown on previous figures) exists between directory branches. An advisor's node is attached to the nodes of all curricula in which he advises. Some advisor files, and hence some student files, might therefore (see Figure 8) be accessible by several paths. The curriculum entry in the general information block of a student's file is



This student can be accessed by the names:
$$220014.50AC1249.00000040
$$220016.50AC1249.00000040
$$220018.50AC1249.00000040
because Jones is an advisor for three curricula.

Figure 8. Interconnections in the ØLF Tree

compared to the current level-two search node to avoid extraneous or redundant processing. Thus, while ØLF files are not organized in a true tree (as defined above), a simple procedure renders the difference inconsequential.

Each search need not be specified from the level-one node. Previously specified nodes are defaulted subject to the conventions that:

1) in any access mode, nodes beneath the level of access become undefined at the completion of the search;
2) all nodes past a changed node must be redefined;
3) whenever a new processor is selected, a new search strategy must be defined starting from the level-one node.

A program can access any file directly (without using the ØLF tree), if its name is known, by invoking the subroutine OPEN. This is not the intended ØLF access mode, and is used in only one instance (see PROCESSORS).

The ØLF tree helps realize the three response-optimizing goals set in SYSTEM CONSIDERATIONS: minimal file accessing, editable-in-core files, and a compact load module.

Batch-access of student files is implemented through systematic progression along the tree. One or more (depending on the access level) directories (where a directory is the internal representation of a node) is referenced repeatedly. In individual access mode, it is not unlikely that files of a common advisor will be processed sequentially. One directory of each level is therefore kept in main memory, and a new directory is loaded only when necessary. Since there is a single level-one file (CURRIC) only levels-two and -three directories are ever replaced.

Like the directory files, the student file has a compact organization. In the prototype design, three directories occupy 2092 bytes while a

student file fills 1648 bytes. These requirements, less than 4K bytes combined, are an acceptable fraction of the 100K allowed to PLØRTS programs. Keeping these files in main memory allows direct access of file items, without jeopardizing the compactness of the load module.

Figure 9 shows the internal representation of the pointer system used in tree access.

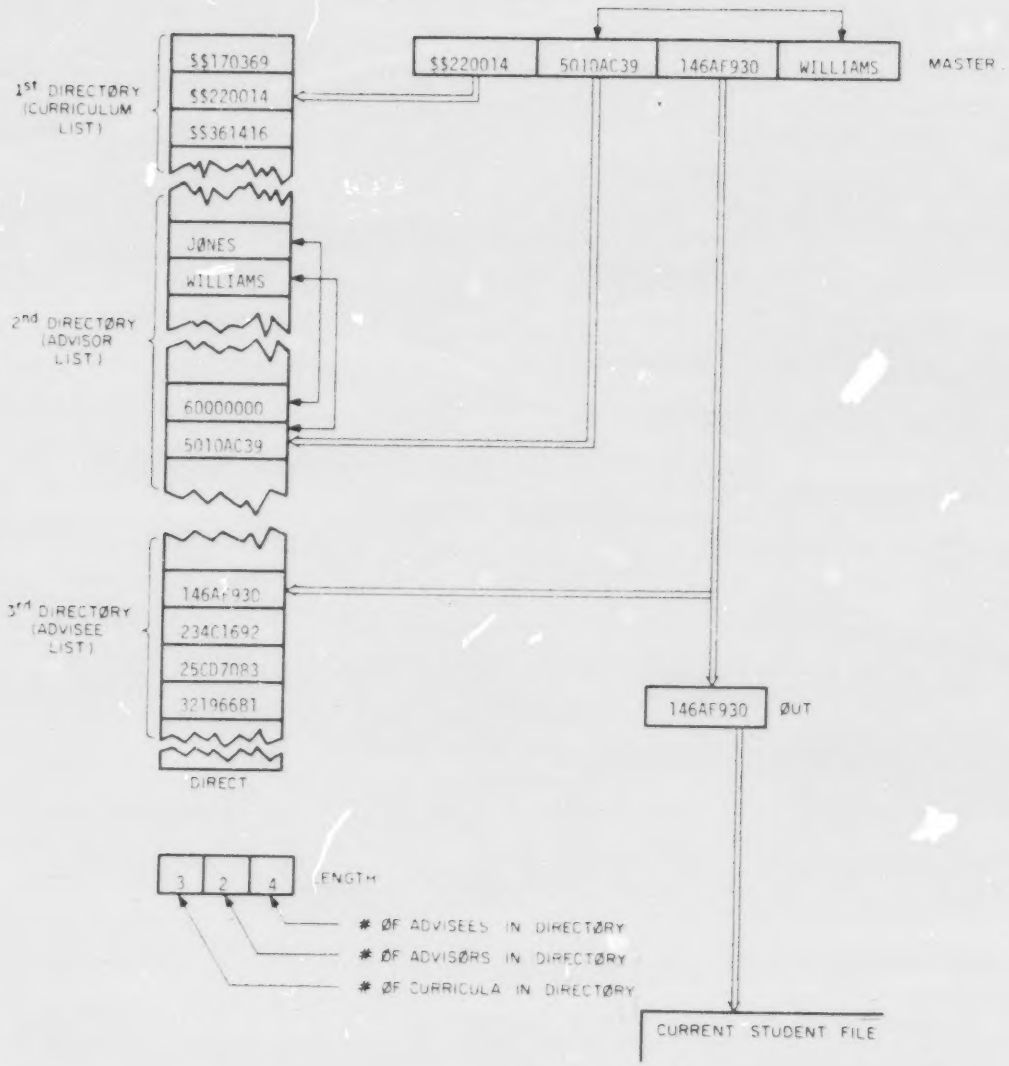


Figure 9. Tree Pointers

4. PROCESSORS

ØLF does not itself manipulate any of the information within individual student files: this is the job of data processing subroutines, hereafter called processors, which operate within ØLF. Each new processor can use previously-written processors as subprocessors. A processor that adds transfer students to ØLF can be constructed from a file creation processor, a program recording processor, and a credit assignment processor (which in turn would invoke a GPA calculating processor).

4.1 The Processor Interface

At the interface between ØLF and the processors are the terminal input buffer and the processor access rights table (PART), shown in Figure 10. The regulated rights are student file I/O, directory file output, and direct access of student files. Access of one processor by another is restricted in only two ways: the subprocessor's file I/O rights must be a subset of the rights of its invoking processor, and no processor can be invoked, directly or indirectly, by itself. The latter restriction is inherent to FØRTRAN; the reasons for the former restriction should become apparent in light of the discussion below.

A necessary condition to guarantee that all instructions intended for ØLF are properly received is the denial of terminal input to the processors. The terminal is, unfortunately, unconditionally available to all PLØRTS programs. After reading enough information to define a search strategy, ØLF rewrites on disc all terminal input until a blank line is submitted. (If input is via the card reader, the first card image not in data format causes buffering to terminate. This card image is used--after return from the active processor--as the next command to ØLF.) The processors are supposed to read

ACCESS RIGHT / PROCESSOR NAME	STUDENT FILE INPUT LIMIT	STUDENT FILE OUTPUT AUTHORIZATION	DIRECTORY-EDITING AUTHORIZATION	DIRECT ACCESS AUTHORIZATION	EXTRA ACCESS AUTHORIZATION
CREATE FILE	1	1	1	1	0
DELETE FILE	0	0	1	0	0
GPA CALCULATOR	1	0	0	0	0
USERA	1	0	0	0	1

The processors named above are described in the section Sample Processors.

Student File Input Limit:
- 0: Read header, general information block
- 1: Read entire student file

Student File Output Authorization:
- 0: Not allowed to rewrite file
- 1: Allowed to rewrite file

Directory-Editing Authorization:
- 0: Not allowed to edit directory
- 1: Allowed to edit directory

Direct Access Authorization:
- 0: Tree access only
- 1: Direct (non-tree) access allowed

Extra Access Authorization:
- 0: Invoke processor once per student file
- 1: Invoke processor once per student file plus once at completion of batch access

Figure 10. Processor Access Rights Table (PART)

from the buffer, but this policy cannot be enforced. A multi-step processor can read an input stream repeatedly by rewinding the sequential disc data set. If necessary, the input stream can be edited for the use of sub-processors.

A file is edited by writing an altered version of the file over the original contents. Write protection for ØLF files is implemented through control of file output. File access is restricted to ØLF since file names are kept secret from the processors. ØLF will not copy a file back onto disc unless the initiating processor has student file output authority in PART. All processors return completion codes; ØLF does not rewrite the current student file--even if the active processor has write authority--if the last completion code indicates a processing error.

Certain processors may use only information found near the beginning of student records. The student file input parameter in PART controls the amount of data read from each record. (The time saved by a partial read-in is less than might be expected--the buffer assigned by the PLØRTS monitor to the disc controller is larger than a complete student file.) Rewriting of a partially loaded file causes erasure of the unread portion of the file; therefore all editing processors (except DELETE FILE) must read entire files.

ØLF access requires reading many directory files. A copy directory parameter in PART tells ØLF whether or not the current level-three directory must be rewritten onto disc before a new one is loaded (equivalent to whether or not the active processor edits the directories). The file insertion/deletion processors, and any multi-step processors which invoke them, must have copy directory enabled.

Statistics-gathering processors may need to be invoked once after the completion of a batch access (e.g., to calculate standard deviations after

determining means). The extra access parameter in PART tells ØLF whether or not the current processor is allowed an extra access.

It should now be clear that the file I/O rights of a subprocessor must be a subset of the rights of its invoker. A subprocessor cannot successfully edit a file if the initiating processor does not have student file output authorization. A subprocessor which requires data from complete student records cannot function correctly if the processor invoking it requests a partial read-in (that is, if the initial processor of a multi-step processor has a limited student file input parameter in PART). A statistics subprocessor may not be able to complete its calculations if the processor using it is not executed once after each batch access.

It is expected that direct file access--file locating without use of the ØLF tree--will be used only by CREATE FILE (a sample processor discussed below) or processors invoking CREATE FILE. For purposes of generality, the direct access capability is included within ØLF and enabled by a direct access parameter in PART.

4.2 Sample Processors

The sample processors discussed below are intended to demonstrate the ease with which new capabilities can be added to ØLF, and to clarify the interface conventions. This section is not an exhaustive list of useful processors.

4.2.1 CREATE FILE (CRFILE)

File creation entails:

1) determining that the proposed file does not yet exist;
2) checking that the directory of the proposed advisor has room for another student;

3) creating the file--with a filled header and initialized general information block, requirements check-off list, and course history block;
4) adding the name of the new file to the disc file of new files (see SAFETY & SECURITY, below);
5) adding the name of the proposed file to the proper level-three directory.

CRFILE is the processor which requires direct file access. The mistyping of SSNs is a predictable error. Standard ØLF (tree) access determines the presence or absence of a student file only from a specific level-three directory. The SSN of a student known to ØLF might be the same as the number mistakenly entered as the SSN of the student for whom the new file is to be created. Unless old and new students both have the same advisor, tree access cannot discover that a file named after the proposed "new" SSN already exists. Opening the old file to create the new student's file destroys the original record.

The SSN of a student is sufficient information to enable ØLF to generate his file's proper name. If tree access fails to find a file with a proposed new name, the ØPEN subroutine is invoked for input. Upon return, if the error flag is set, the proposed file name is unknown to ØLF (and PLØRTS), and the proposed file can be created. If the file already exists, the header is read, determining to whom ØLF believes that SSN belongs. (Both forms of access are performed by ØLF, not CRFILE.) The distinguishability of student and advisor file names (see File Names, above) prevents attempting to read a student header from an advisor file, which would cause a data format error.

4.2.2 DELETE FILE (DLFILE)

File deletion entails:

1) determining that the to-be-deleted file exists;

2) removing the name of the to-be-deleted file from the proper level-three directory;

3) entering the name of the file to be deleted into the deallocated files list;

4) commenting when an advisor loses his last advisee.

4.2.3 GPA CALCULATOR (CALC)

CALC finds the GPA resulting from practically any subset of courses taken by any student. The selectable subsets are the intersections of any of the following sets:

1) all courses within the major and minor subjects;

2) all courses within the major subject only;

3) all courses within the minor subject only;

4) all courses within a named department;

5) all courses taken during and after a specified semester;

6) all courses taken during or after a given status (e.g., freshman, sophomore, etc.) is reached;

7) all courses within the last given number of hours;

8) all courses including and above a specific level (100, 200, etc.);

9) all courses not yet included in the cumulative GPA (grades just recorded).

CALC returns, in addition to a GPA, the number and total hours of the courses used in the calculation.

4.2.4 A SAMPLE STATISTICS PROCESSOR (USERA)

Most user-supplied processors are expected to be statistics-gatherers. The sample processor described below is intended to be indicative of the

potentialities of ØLF; it also, by invoking CALC, illustrates the rules governing multi-step processors (see their respective PART entries, in Figure 10).

The XX department wants to prove or disprove the belief of some of its advisors that the course YYY108 should be a prerequisite for all 200- (and above) level XX courses. The advisors who feel that YYY108 is a useful course have consistently recommended it to their advisees. This processor will calculate and compare the GPAs in 200- and above level XX courses for the two sets of XX students: those who took YYY108 as suggested, and those who did not.

The department name (XX), the course name (YYY108), and the course cutoff level (200) are read in by USERA. The input line indicating the course cutoff level is passed by USERA, via the input buffer, to CALC. USERA consequently can be used by any department interested in any course; any set of courses which can be selected by CALC can be examined by USERA.

5. SAFETY & SECURITY

Two crucial problems of automated system design are reliability (will the system be available when needed) and safety (is the system protected against accidental and malicious tampering). ØLF, as a program package within PLØRTS, can be no more dependable or secure than PLØRTS limitations allow.

ØLF is available whenever PLØRTS is available. (A TSBATCH job can execute ØLF programs when the PDP-7 (the terminal--IBM 360 interface) is not running, but the benefits of interactive processing are lost.) ØLF clearly must be paralleled by an alternate system at least at the times of heaviest use--the registration and advanced enrollment periods. Printouts of the ØLF student records can supply such a supportive system. In most cases, advisors will be able to counsel using only the information on such printouts. In the event of an equipment failure, only the printouts will be available--but this is the situation under the present record keeping system. A special processor can be added to ØLF to furnish a conveniently formatted record from each student file.

The protection of program files is straightforward. Full access to any PLØRTS file requires knowledge of the triplet (PS#,username,filename). Each user is forced to use his own copy of ØLF (and its current processors) by keeping the username of a master program file secret. As a secondary benefit, this procedure prevents user-supplied processors of narrow applicability from proliferating in the master program file. The minor editing of ØLF necessary before use of a new processor is made on an ØLF copy, and, if the processor is to be kept, the master copy can be updated by authorized personnel after debugging is complete. Extra copies should be destroyed after use. A backup

tape copy of the master program file (inaccessible without the tape's name) is kept since a user who learns the codeword might maliciously alter the program file.

Data files must be protected from undebugged processors and PLØRTS (hardware and system software) failures. Processor-inflicted damage is possible only when a processor does not conform to ØLF interfacing conventions. A processor with PART entries for both partial student file read-in and student file output authorization may erase parts of files; a processor with PART student file output authorization that returns an undeserved no-error completion code may overwrite correct information; a directory editing processor without PART authorization for directory file output can render groups of student files inaccessible by normal ØLF methods. Processors can be debugged with a single student file--preferably of an imaginary student. A backup tape of all data files should be prepared after each execution of ØLF (as another job step) in case of errors in the next execution.

ØLF prints a message immediately before and after invoking a processor. A missing completion message indicates which file was being processed when a system failure occurred. A more serious problem is the state of the directory files following a system failure. DLFILE maintains a file on disc of all student files edited from level-three directories but not yet deallocated. If DLFILE was the active processor, the last file deleted from the level-three directory loaded at the time of failure may not have been added to the deallocate list. CRFILE maintains a similar list of all student files added to level-three directories during the current execution. If CRFILE was the active processor at failure, the most recently created file might not have been added to the list. In all cases, the output of ØLF indicates which directory and student file were loaded when failure occurred.

If failure occurs, any insertions and deletions to the current level-three directory by CRFILE and DLFILE may not be reflected in the disc copy of this directory. An error recovery program (see UTILITIES PROGRAMS) revises the directories to reflect the changes shown in the create and deallocate lists. Any system failure (except a head crash that destroys one of the recovery files at a time when CRFILE or DLFILE is executing) is recoverable.

As programmed, ØLF data files are not safe from malicious individuals. Files are accessible for proofreading and/or simple editing by explicit terminal command because all file names are constructed of characters available on terminals. File security is thus diminished in return for ease of access. The remainder of this section discusses possible changes to make ØLF more secure at the expense of convenience.

Files whose names contain "characters" not included in the terminal character set are accessible only by program. A one-to-one substitution of non-printable for printable characters--a simple table look-up procedure--suffices to adapt the present file name generating subroutine to this naming convention. Malicious mischief immediately requires a higher level of effort.

Added safety could be obtained by keeping the substitution table in a disc file protected by a different codeword than the master program file. If security is violated, a recovery procedure would be: change the contents of the substitution table; change the codeword guarding the substitution table; rename, according to the new table (via utility program), all data files.

6. UTILITIES PROGRAMS

ØLF must be supported by several programs which are not needed by general users.

CRFILE and DLFILE edit the level-three directories; the program DIRECTORY EDITOR (DREDIT) adds or deletes advisors and curricula in the higher-level directories. DREDIT is similar to the other file editors, and is kept apart from ØLF only because its relatively infrequent use does not justify an increase of ØLF storage requirements.

An error recovery program (RECOVR) is used after a PLØRTS failure occurs following use of CRFILE, DLFILE, or DREDIT. RECOVR edits directories to correspond to the create and deallocate lists maintained by these programs. The file-copying program (CØPY) and file-restoring program (RECØPY) safeguard all student records.

Envisioned program-approving and progress-assessing processors will use various course and prerequisite data files. These files will require programs for their creation/deletion/editing, which in turn might need error recovery programs. Like DREDIT, the latter will be used too infrequently to incorporate in ØLF proper, and will undoubtedly be run as independent programs.

7. CONCLUSION

An on-line filing (ØLF) system for student records has been designed and programmed. ØLF can be instructed to access various groups of student records for use by various data processing subroutines (processors). Since all records are stored on-line and are machine readable, ØLF provides a statistics-compiling capability. It is intended that ØLF eventually be extended, by the programming of new processors, into a complete advisory and record-keeping system.

APPENDIX A

COMPUTING SERVICES OFFICE DOCUMENTS RELEVANT TO ØLF

This appendix contains reproductions of four documents pertinent to the use of PLØRTS and/or CALL/360-ØS FØRTRAN. Typographical errors present in the original Computing Services Office documents have been corrected. Certain conventions observed in the body of this report differ from conventions observed in the following documents.

COMPUTING SERVICES OFFICE PLORTS FORTRAN H 17-1
UNIVERSITY OF ILLINOIS at URBANA-CHAMPAIGN

FORTRAN is a widely used programming language, implemented as part of the PLORTS timesharing system. To use PLORTS FORTRAN, a person must be authorized to use the PLORTS system, and must have use of a Teletype-compatible or IBM 2741-compatible device. The PLORTS system is discussed in manual CSO-1000, which is available through the consultants, room 166 DCL, and room 83 Commerce West, or for sale at the Illini Union Bookstore.

The FORTRAN language implemented is described in IBM manual H20-0710, CALL 360-OS FORTRAN LANGUAGE REFERENCE MANUAL, except that CALL/OS commands (such as FILE) are not implemented in PLORTS, and the maximum record that may be read or written using formatted I/O is 80 characters. A procedure to aid in converting existing programs from OS/360 FORTRAN to PLORTS FORTRAN, called FORTCONV, is described in a separate reference guide.

Once a FORTRAN program has been entered into a PLORTS file, the program may be executed interactively by typing FORTRAN. Execution of a FORTRAN program may be stopped by execution of a STOP statement, discovery of a run-time error, or by typing KILLL if the program is waiting for input. Execution of a FORTRAN program is halted every 15 seconds and the user is asked if he or she wishes to continue. To continue, type YES or Y; to stop, type NO, N or KILLL, e.g.,

```
   1.000 "  THIS PROGRAM COMPUTES MEAN, VARIANCE AND STANDARD
   2.000 "  DEVIATION FOR A SEQUENCE OF POSITIVE NUMBERS AND
   3.000 "  SAVES THE NUMBERS IN A PLORTS FILE NAMED SAVEA.
   4.000 CALL OPEN(1,'SAVEA','OUTPUT')
   5.000 1 READ(5,*) A
   6.000 IF(A.LT.0.0) GOTO 3
   7.000 WRITE (1,2) A
   8.000 2 FORMAT (E15.6)
   9.000 I = I + 1
  10.000 S - S + A
  11.000 Q = Q + A*A
  12.000 GO TO 1
  13.000 3 S = S/I
  14.000 Q = Q/I - S*S
  15.000 A = SQRT(Q)
  16.000 WRITE(6,4) S,Q,A
  17.000 4 FORMAT(14X,'MEAN',14X,'VARIANCE',7X,%
  18.000 'STANDARD DEVIATION'/3F20.6)
  19.000 STOP
  20.000 END
FORTRAN
?1
?2.0
?3E0
?-1
          MEAN                VARIANCE          STANDARD DEVIATION
      2.000000              0.666666              0.816496
```

FORTCONV is a procedure to aid in the conversion of programs from IBM FORTRAN to PLORTS FORTRAN.

The program accepts a FORTRAN source program, and outputs a PLORTS FORTRAN version. If conversion is needed and is simple, (for example, changing the C in a comment to a "), statements are converted. If conversion is more difficult, or would require interpretation of the program, the statement is flagged and deleted from the output.

Suggested use of the program:

```
// EXEC FORTCØNV

{FORTRAN program(s)}

/*
```

The output from this run may be inspected and corrections made. When an acceptable PLORTS FORTRAN deck is produced, it may be checked and filed at the same time by:

```
// EXEC FORTCØNV
PS=ps#,NAME=signonname,DEST=filename

{FORTRAN program}

/*
```

NOTE: The program will not insert CALL ØPEN or CALL CLØSE statements. The printed output from the program is relatively lengthy. Use a lines estimate on your ID card approximately equal to twice the number of input cards plus 200.

A job card (green card) and your ID cards (see REF-01) must be at the front of every deck.

COMPUTING SERVICES OFFICE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

360 Notice No. 1676

CALL/OS Load Modules

The PLORTS CALL/OS system has been extended to allow for the use of CALL/OS load modules. The object code generated from a FORTRAN or BASIC compilation can be saved in another file by using the parameter 'FILE=filename' on the FORTRAN or BASIC command.

Examples:

```
FORTRAN    FILE=OBJECT1[,other parms]
BASIC      FILE=OBJECT2[,other parms]
```

The object code from the compilation will be stored in the file specified. The original filename will become a catalogue file, making normal PLORTS editing of the object code impossible. Note, this object code is only valid to run on the PLORTS CALL/OS system.

In order to run an object program, issue an 'EXEC' or 'EXECE' command from closed mode. 'EXECE' like 'OPENE' allows access to files under another PS and name.

Examples:

```
EXEC       OBJECT1[,other parms]
EXECE      9999.SIGNON.OBJECT1[,other parms]
```

Note, a BASIC program can not be executed with the DEBUG option unless it was compiled with DEBUG at the time the object code was saved.

The examples above indicate that parameters can be used to alter the compilation and execution of CALL/OS programs. The following parameters are currently available.

FILE=filename (described above)

DEBUG Allows post-mortem analysis, correction and restarting of BASIC programs. See OPENMSG 7 for a complete description.

TIME=dd where 0<dd<100 dd controls the frequency with which 'CONTINUE?' prompts are issued. An executing program may use a minimum of dd seconds of CPU time between successive prompts. Default value is 5 sec.

In addition, CALL/OS FORTRAN file I/O has been extended to allow data to be filed at the end of an already existing file. To use this facility simply open a file for 'MOD' instead of 'OUTPUT'.

```
CALL OPEN (8,'FILENAME','MOD')
```

This feature is valid for both formatted and unformatted files. Opening a file for 'OUTPUT' will continue to destroy and recreate the file.

David Subject
March 5, 1973

DS:jmv

COMPUTING SERVICES OFFICE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

360 Notice No. 1697

Modifications to CALL/OS

The filing of CALL/OS object code (Notice No. 1676) has been modified to eliminate the saving of certain unnecessary data, thus reducing the amount of code saved and the number of blocks required to save it. The modification will only effect programs using large non-initialized arrays; that is, arrays not initialized during compilation such as with a FORTRAN "DATA" statement. Existing object modules may still be executed, but must be recreated to make use of the new feature.

In addition, the CALL/OS FORTRAN "OPEN" subroutine has been modified to allow an optional 4th parameter in which a completion code can be returned. This feature will allow a program to intercept and interpret any errors which occurred trying to open a file and continue executing if desired. If the 4th parameter is not specified then any errors which occur will terminate execution as is now the case. The 4th parameter must be an integer*4 variable or member of an integer*4 array.

Examples:

```
CALL OPEN (8,'FILE NAME','INPUT',I)
```

Possible completion codes are:

0 - successful open
1 - no space available for an 'OUTPUT'file
2 - 'INPUT' file does not exist
3 - 'INPUT' file is a catalog file
4 - 'OUTPUT' file is currently being used
5 - an illegal file name was specified

If the return code is 0, the file is open. In all other cases, it is not.

Dave Subject
April 11, 1973

DS:jmv

APPENDIX B

ØLF OPERATIONS MANUAL

B.1 Introduction

This appendix is a self-contained, non-technical manual for ØLF users. ØLF is an on-line filing and data processing system for student records. It consists of a data retrieval program and data processing subroutines (processors). Search commands and processor data statements form the ØLF instruction set.

ØLF input records are limited in length (by the terminal line capacity) to seventy-two characters. An input record is divided into eight fields to form an ØLF instruction. A new field begins every tenth character, starting with the left-most character of the input record. Fields one through eight are numbered consecutively from left to right. The contents of each field are left-justified within that field. Formatted input does not inconvenience an ØLF user since PLØRTS terminals can be programmed to tabulate.

B.2 Search Commands

A processor and a set of student records are selected by search commands to initiate the processing of information. A processor is specified by a single LØCATE statement; various sets of student records are selected by from one to three LØCATE statements, as explained below. All forms of search commands are shown symbolically in Figure 11, where the decomposition of each command into fields is indicated. The symbols in field three are replaced as described below to form syntactically correct ØLF instructions.

```
 Field--1  Field--2  Field--3
+........++........++........+
LØCATE    PRØCESSOR procname
LØCATE    CRRCLM    curr#
LØCATE    ADVISØR   advname
LØCATE    STUDENT   ssn
```

Figure 11. Search Commands

A processor is selected by a single LØCATE PRØCESSOR command. The desired processor's name, of eight or fewer letters in length, is substituted for the symbol "procname" shown in Figure 11. A new set of student records must be selected after each processor request.

The records of the set of all students enrolled in a specific curriculum are selected by a LØCATE CRRCLM command. A six-digit curriculum code replaces the symbol "curr#" of Figure 11.

The records of all students sharing a common advisor and enrolled in a previously named curriculum are selected by a LØCATE ADVISØR command. The name of the common advisor, of eight or fewer letters in length (truncated if necessary), replaces the symbol "advname" shown in Figure 11.

The record of one student counseled by a previously named advisor and enrolled in a previously specified curriculum can be selected by a LØCATE STUDENT command. The student's nine digit social security number (SSN) is substituted for the symbol "ssn" shown in Figure 11.

The LØCATE CRRCLM, LØCATE ADVISØR, and LØCATE STUDENT commands, as described above, are used in conjunction to select single student records. Individual access mode is convenient when separate data is needed to process each file (e.g., when recording course grades). When the data to be processed is present in the student records before processing begins (e.g., when compiling curriculum-wide statistics from the completed coursework of students), a

batch-mode access of records--access of many records by only a few commands--is useful.

Alternate forms of the non-processor search commands request three levels of batch access. The symbols "curr#", "advname", and "ssn" of Figure 11 take the value "*" (asterisk) to request access, respectively, to: all records known to ØLF (the records of all students in all curricula), the records of all students previously specified in a LØCATE CRRCLM statement, and the records of all students previously specified by LØCATE CRRCLM and LØCATE ADVISØR statements. These search strategies are called, respectively, batch access levels zero, one, and two. For example, the commands

```
LØCATE    CRRCLM    220014
LØCATE    ADVISØR   SMITH
LØCATE    STUDENT   *
```

select for processing, at batch-access level two, all advisees of Professor Smith in curriculum 220014. A search command containing an asterisk completes the definition of the record set to be processed.

B.3 Data Statements

As explained above, individual access mode is used primarily by processors which read data for each student file accessed; batch-access mode is used primarily by processors which use the information already in the student records. Two data statement formats correspond to the two access methods.

A data statement which is to be read by an individual-access processor is labeled with the SSN of the student whose record is to be processed. The label is compared to the SSN of the owner of the last-requested student record to minimize erroneous input. The SSN occupies the first field of the input record; the remaining seven fields are filled to satisfy the information requirements of the particular processor.

The data read by batch-access processors relates to groups of student records. The word DATA in the first field of the data statement identifies a batch-access data statement. The information required by a processor is located in the remaining seven fields of the statement.

ØLF search commands occupy only three of the eight fields in each statement. A processor (e.g., DLFILE, discussed below) can be written tc extract information from fields four through eight of the last search command in a file specification. The data fields of a search command can be used in addition to standard data statements. Figure 12 illustrates the general forms of data statements.

```
 Field--1  Field--2  Field--3  Field--4  ... Field--8
↓........↓↓........↓↓........↓↓........↓     ↓↓

333224444 Datum 1   Datum 2   Datum 3    ... Datum 7
DATA      Datum 1   Datum 2   Datum 3        Datum 7
LØCATE    Filetype  Filename  Datum 1        Datum 5
```

The third statement is a combined search command/data statement. If "Filetype" is not "STUDENT" then "Filename" must be "*" (i.e., data may appear only on the last of a set of search commands).

Figure 12. Data Statements

B.4 Errors

ØLF language errors can be grouped into three classes: processor errors, non-translatable search commands, and invalid search commands. A secondary classification groups errors into recoverable and non-recoverable categories: all errors except non-translatable search commands are recoverable.

The occurrence of a non-translatable or an inconsistent data field on a data statement constitutes a processor error. (The presence or absence of information in the data fields of a search command is determined after the

selection of an access mode; therefore, a processor error cannot be caused by an error in a data field of a search command unless the command fields are error-free.) A processor responds to an error by printing an error message and either skipping the processing step or performing only that portion of the processing step which is independent of the improper field. The particular response depends on the processor involved.

A request for an unknown student or directory file constitutes an invalid search command. When ØLF detects such a command, an error message is printed and a new search command is requested.

Another form of invalid search command is possible when CRFILE (i.e., the file-creating processor, discussed below), or a processor invoking it, is active. ØLF ordinarily searches for a requested student record only in the file of his advisor; when a new file is to be created, ØLF must determine that the proposed file does not exist in the file of any advisor. An attempt to access a file by a general search should be unsuccessful. Discovery of a file by general search is an error to which ØLF responds by printing an error message and a new search request.

A non-translatable search command error can take two forms: a syntactically incorrect statement, or a statement that is out of context. ØLF syntax has been explained above.

ØLF expects to receive commands according to the hierarchy of Figure 13. An out-of-order statement is considered a non-translatable search command error. A statement may be out of context because another statement is missing. A long series of data statements could result from omitted search commands; accordingly, five or more consecutive data statements are interpreted as a missing search command--a form of non-translatable search command error.

The recoverable/non-recoverable classification of errors relates to input mode and is discussed in the next section.

Level	Statement
1	LØCATE PRØCESSØR
2	LØCATE CRRCLM
3	LØCATE ADVISØR[1]
4	LØCATE STUDENT[1,2]
5	Data Statements[3]

---

[1] Omitted when preceded by LØCATE CRRCLM *

[2] Omitted when preceded by LØCATE ADVISØR *

[3] Data statements are required by processors, not by ØLF. ØLF enforces a limit, discussed below, of four consecutive data statements.

Figure 13. ØLF Statement Hierarchy

B.5 Input Modes

ØLF was designed for interactive use. Interaction, unfortunately, is very time-consuming when large amounts of information are being read or printed. The ØLF user can choose card (off-line), rather than terminal, input mode in such cases.

ØLF treats the two input modes differently in three significant areas: printing, input stream delimiting, and response to errors. All card input is printed to make possible the evaluation of ØLF output; terminal input is not printed since the user can read the input which he himself has typed.

In terminal mode, the stream of data statements is delimited by a blank line (i.e., by typing carriage-return as the first character of an input record). A non-blank line which cannot be recognized as a data statement is an error. It is unreasonable to require that blank cards be inserted throughout an input deck. In card mode, the first card in the data stream which, by error or design, cannot be recognized as a data statement terminates the stream. On the assumption that the first statement following data is a

search command, the image of this card is retained for interpretation as a command.

Any error detected by ØLF in terminal mode can be corrected by the user. In card mode, ØLF terminates processing upon discovery of non-recoverable errors. A non-recoverable error, as defined above, is a non-translatable, out-of-context, or missing search command. An unavailable command might be a LØCATE PRØCESSØR statement; continued processing following such an error would jeopardize the contents of the student records.

It should be clear that card mode has both advantages and disadvantages over terminal mode. The user can have keypunchers prepare his input deck, and his job can be run without the need of his periodic intervention; an error, either on his part or on the part of keypunchers, may waste a run. Certain jobs can best be done off-line (e.g., printing a copy of each student record for back-up purposes); jobs containing many search commands may best be run interactively.

B.6 Use of ØLF

A user must log into PLØRTS to run ØLF. The log-in procedure is described in the PLØRTS Guide (reference 4). When card mode is to be used, log-in should be accomplished through the catalogued procedure TSBATCH.

Processing is initiated by an EXEC command (see Appendix A, 360 Notice No. 1676). ØLF will type

INPUT ON CARDS--TYPE YES OR NO?

Terminal users should type NO; off-line users should respond (by a card from the input deck) YES.

Once the input mode has been determined, ØLF will type

TYPE PRØCESSØR REQUEST

?

to which the user should respond with a LØCATE PRØCESSØR command. In card mode, an error at this point causes termination; in terminal mode, an error causes a processor request to be printed. This is standard ØLF response to a non-recoverable error.

Once the processor request has been satisfied, ØLF types

TYPE SEARCH REQUEST

?

to which the user should respond with an appropriate search command. Certain errors at this point are non-recoverable (see Errors, above). If the search command is acceptable, but does not constitute a complete search definition (e.g., LØCATE CRRCLM 220014), another search command is requested.

Once the search strategy is set, ØLF attempts to access the file(s) specified. At this point ØLF determines whether or not the curricula and advisors specified exist. Naming an unknown curriculum or advisor in a search command is a non-recoverable error.

Another error may occur when CRFILE, or a processor invoking it, is active. The existence of a proposed new file outside of the directory of the proposed advisor is noted by an error message, and the processing step involving that student file is skipped. This is the standard ØLF response to recoverable errors.

It is a recoverable error in most cases to specify a non-existent student record; when CRFILE or a processor invoking it, is active, the request of a non-existent file is not an error.

Once a student file has been accessed, or if it is to be created, once it has been found not to exist, ØLF types

TYPE DATA (IF ANY)

?

to which the user should respond as appropriate for the active processor. Some processors may expect data from the last LØCATE statement in place of, or in addition to, any data submitted at this time. ØLF tests the data statement for the proper format of the current access level. Out-of-format statements cause a warning to be printed and, in terminal mode, the data request to be reprinted. Data requests are repeated until a data delimiter or a fifth data statement is detected. Five consecutive data statements constitute a non-recoverable error.

The user will receive data requests before each processing step when in individual access mode. During each batch access, data is requested only once, immediately before processing the first of the set of selected files.

Prior to entering the active processor, ØLF types a message naming the current processor and the student file to be processed. A completion message is printed upon return from the processor. Processor errors are treated differently by various processors, but are always recoverable. When an error occurs, the unedited copy of the current student record is kept.

Following an error-free processing step, certain processors require ØLF to edit its own files. ØLF suppresses editing when an error arises while attempting to edit.

When a just-completed processing step involves the last of the selected student records (as is always the case when in individual access mode), ØLF requests a new search command, and the procedure repeats.[1] If the last-named curriculum and/or advisor describes the next set of student records, then new search instructions can begin, respectively, with a LØCATE ADVISØR or LØCATE STUDENT command.

---

[1]Certain processors (e.g., USERA, below) are invoked once for each student record accessed plus once at the completion of the batch access.

When a completed processing step, or a processing step which was skipped because of a recoverable error, did not use the last of the selected set of student records, then (attempted) processing of the next file begins. This case occurs only when in a batch-access mode.

After completion of all processing requested by a set of commands (which may consist of many processing steps when in a batch-access mode), an alternate response to ØLF's search request is a LØCATE PRØCESSOR command. Following such a response, a new set of search commands is required, starting with LØCATE CRRCLM.

The user, upon finishing all processing, can signal completion as a response to either a processor search request by typing DØNE. ØLF responds

AFTER THIS JOB ENDS, TYPE: RUNN CØPY.

PLØRTS signals completion of a job by printing STØP, and the charges for all computer time used. Entered after the ØLF job is completed, the RUNN CØPY command initiates another program which prepares a tape copy of ØLF files for use in the event of a future PLØRTS failure. PLØRTS will respond by typing a job receipt. The user may log-out after initiating CØPY.

B.7 Present Processors

As of this writing, ØLF includes five processors. A brief description of each processor and its data requirements is given below.

DUD is a processor which performs no processing. It demonstrates safely to new users the order in which student records are processed in the various access modes. Any input submitted to DUD is ignored.

CRFILE is the processor which creates student files. The LØCATE STUDENT statement specifying the to-be-created file must include the following

data items: the student's name (last name first) in fields four and five; the proposed entry semester in field six; the proposed entry year in field seven.

Another data statement is used when the student is enrolling in a co-curriculum: the characters CØCRRLM belong in field two and the co-curriculum's six-digit code belongs in field three. Any additional data statements are ignored.

CRFILE signals (via its completion code) an error in processing if a datum on the LØCATE card is non-translatable, or if the proposed entry date is in the past. A non-translatable or unknown co-curriculum code causes processing of the co-curriculum to be skipped but does not prevent file creation. An appropriate message is printed by CRFILE in each of the above cases.

If the data on the LØCATE statement is acceptable, the new file is created, containing the student's SSN, name, college code, curriculum code, advisor's name, and entry date. When submitted correctly, a co-college code and co-curriculum code are also stored. The remainder of the record is initialized as appropriate.

ØLF discards the new file if CRFILE signals an error. When processing goes normally, ØLF attempts to add the name of the new file to its master files (directories). An error message is printed and the new file is discarded if the directory of the student's advisor is full.

DLFILE is the processor which destroys existing student files. Fields four and five of the LØCATE STUDENT card which specifies the to-be-destroyed file must contain the name of the owner of that file, exactly as stored in the file. This comparison prevents the accidental destruction of files. The error message printed when an incorrect name is supplied shows the owner's name, so that misspelling can be differentiated from more serious mistakes.

If the name field on the LØCATE statement matches the name stored in the student file then normal completion is signalled to ØLF, which removes the name of the file from the appropriate directory. If the student file was the only file named in the directory, a message is printed identifying the now-empty directory.

CALC is the processor which calculates grade-point averages (GPAs). In addition to a GPA, the number and total hours of courses used in the calculation are available.

CALC will find the GPA of practically any subset of courses in a student record. When no data statements are submitted (CALC does not look for data on a LØCATE statement), the GPA is found for courses not yet in the cumulative GPA--presumably the last semester's grades. The cumulative GPAs of all courses, major area of study, and individual departments are also updated.

Combinations of data statements request GPA calculations on different subsets of the completed coursework. Any group of four or fewer (ØLF limitation) of the following statements may be used to choose a particular set of courses. The GPA calculation is done with the set of courses described by the intersection of the sets selected by the individual statements.

A type of course (major or minor) is selected by a data statement with TYPE in field two and MAJØR or MINØR in field three. Both major and minor areas of study are selected by using two data statements. Alternatively, MAJØR and MINØR can appear in fields three and four of a single statement (either word first).

Courses of one department are selected by a data statement with DEPT in field two and the five-letter department code in field three.

Courses completed since a given date (semester) are chosen by a data statement with SEMESTER in field two, a semester name--FALL (or AUTUMN), SUMMER, or SPRING--in field three, and the year in field four.

Courses completed since attaining a given academic status are selected by a data statement with STATUS in field two and an academic year--FRESHMAN, SØPHØMØRE, JUNIØR, or SENIØR--in field three.

The last n hours of coursework completed are chosen by a data statement with HØURS in field two and a non-negative integer n in field three. The value of n must be written in three digits, using non-significant zeroes as needed.

Courses at and above a given level n (e.g., 100, 200, etc.) are selected by a data statement with LEVEL in field two and a positive integer n in field three. The value of n must be written in three digits, using non-significant zeroes as needed.

Figure 14 shows typical CALC data statements.

```
 Field--1  Field--2  Field--3  Field--4
↓.........↓↓.........↓↓.........↓↓.........↓

333224444 TYPE      MAJØR     MINØR
333224444 LEVEL     200
333224444 HØURS     030
```

The above three statements, if submitted to CALC, would request a GPA calculation of all advanced level (i.e., 200-level and above) courses within the last thirty completed hours of study and within either the major or the minor areas of study for the student whose SSN is 333-22-4444. Notice that the number thirty (third statement) is written with three digits.

Figure 14. Sample CALC Data Statements

USERA is a statistics-compiling processor which helps determine the value of a course of interest as a prerequisite for coursework in any chosen department. The GPAs of students who have taken the course are compared to the GPAs of students who have not taken it.

The name of the course and department under consideration are submitted to USERA, respectively, in fields four and five of a LØCATE statement. Specific courses from within the chosen department are selected for the comparison by CALC-form data statements, which are used by CALC. USERA generates the data statement for CALC which selects the chosen department.

Figure 15 shows statements required for a typical use of USERA.

```
 Field--1  Field--2  Field--3  Field--4  Field--5
+........+.........+........+........+........+

LØCATE    PRØCESSØR USERA
LØCATE    CRRCLM    220014
LØCATE    ADVISØR   *         XX        YYY108
DATA      STATUS    SENIØR
DATA      LEVEL     300

The above statements command ØLF to compare GPAs of
two groups of senior students enrolled in curriculum
220014--those who have taken the course YYY108, and
those who have not.  Only 300-level XX courses are
used in the calculations.
```

Figure 15. Sample Use of USERA

APPENDIX C

NEW PROCESSORS

The capabilities of ØLF can be increased, when desirable, by the addition to ØLF of new processors.[1] This appendix is concerned with the mechanics of creating new processors. Variable names and statement numbers used in the discussion below appear in APPENDIX D: ØLF PROGRAM LISTINGS.

C.1 Changes in ØLF

A few minor changes are made in ØLF to accommodate each new processor. In combination with reference to the program listings, the following remarks should make clear the nature and necessity of these modifications.

1) The number of processors assumed to exist is given by the initial value of the INTEGER*4 variable RØWS.
2) There should be one row in the INTEGER*4 array PART for each processor. The PART entries of each processor are stored in one record (card image) of the disc file named PART, according to a FØRMAT of (5(I1,1X)).
3) The name of each processor is stored as the value of an element of the REAL*8 vector PRØCS.
4) Each processor may be invoked by only a single CALL statement, which in turn may be reached only via the computed GØ TØ at statement 569.
5) Each processor must accept the INTEGER*4 argument ERRØR. ERRØR is the completion code of the processor:

---

[1] System expansion is restricted only by the 100K byte main memory limitation.

a value of zero implies error-free processing; a value of one indicates a processing error.

C.2 Scratch Files

Some processors will use disc files to save intermediate results. A scratch file must not be given the same name as any present or potential ØLF file. The file-naming conventions have been discussed in the text of this report. Program files and a few data files have names selected without regard to any specific convention. These files are named: ØLF, DREDIT, RECØVR, CØPY, RECØPY, BUFFER, PART, CURRIC, DATES, CØPY1, CØPY2, SAVE, CCØPY, RESTØRE, DELETE1, DELETE2, CRLIST, and DLLIST. Any other names may be used to identify scratch files.

C.3 Programming Aids

The task of programming an ØLF processor is simplified by the modular structure and debug option of ØLF. Several character-manipulation and file-handling subprograms compose a large portion of ØLF's code. With the exception of a subroutine which prints ØLF error messages, each subprogram was written with a far broader capability than ØLF required. Each subprogram and processor contains a set of debug print statements which may be enabled (disabled) by the initialization of the INTEGER*4 variable DEBUG to one (zero). Utilization of ØLF components thus simplifies the debugging of new processors while minimizing the amount of new code to be debugged.[1]

C.4 Information

ØLF can communicate large amounts of information to its processors.

---

[1] Use of existent subprograms, which minimizes storage requirements, is also desirable in view of the main memory limitation.

Additional information can come from user input and data files. This section discusses the nature of this information.

Most of the information that ØLF will share with its processors is found in CØMMØN. This information consists of a student file (HEADER through DBLØCK) and copies of directories and directory-related information (VEC through LL). Additional information may be communicated through a processor parameter list. The mandatory completion code (ERRØR) was discussed above. FINI, an INTEGER*4 variable whose value changes from zero to one upon the completion of a batch access, tells a processor with extra access authorization in PART (e.g., USERA) whether or not a certain execution of the processor is the extra access.

ØLF accepts a maximum of four consecutive data statements (exclusive of data on a search instruction). User input is stored in the disc file BUFFER. A processor which edits BUFFER for its subprocessor(s) need not observe the four statement limit.

The remaining sources of information are envisioned system data files. The anticipated expansion of ØLF into a full advisory system will necessitate the creation of such data files as course equivalence tables and curriculum requirements lists. A program-approving processor would logically read from the former; a progress-assessing processor would necessarily use the latter.

APPENDIX D

ØLF PROGRAM LISTINGS

```
"       THE OLF PROGRAM PACKAGE
"
"       CLF MAIN PROGRAM
"
"       VECTOR CONTAINS THE 3 DIRECTORIES:
"         VECTOR(1) TO VECTOR(200) HOLD CURRIC (LEVEL-1)
"         VECTOR(201) TO VECTOR(240) HOLD A LEVEL-2 DIRECTORY
"         VECTOR(241) TO VECTOR(260) HOLD A LEVEL-3 DIRECTORY
"       MASTER IS THE ARRAY OF SEARCH NODES
"         MASTER(1) TO MASTER(3) HOLD THE 3 SEARCH NODES
"         MASTER(4) HOLDS AN ADVISOR NAME--RELATED TO MASTER(2)
"       DUPLIC IS A COPY OF MASTER
"       LAST(1) TO LAST(3) HOLD THE NAMES OF LOADED DIRECTORIES
"       PROCS HOLDS THE NAMES OF ALL (ROWS) AVAILABLE PROCESSORS
"       PART IS THE PROCESSOR ACCESS RIGHTS TABLE:
"         PART HAS SIZE ROWS X 5--SO  PROCESSORS CAN BE ADDED
"         ROW I HOLDS THE RIGHTS OF THE I-TH PROCESSOR
"         COLUMN 1 IS STUDENT FILE INPUT AUTHORIZATION
"         COLUMN 2 IS STUDENT FILE OUTPUT AUTHORIZATION
"         COLUMN 3 IS DIRECTORY FILE OUTPUT AUTHORIZATION
"         COLUMN 4 IS DIRECT ACCESS AUTHORIZATION
"         COLUMN 5 IS EXTRA PROCESSOR CALL AUTHORIZATION
"       THE STUDENT FILE IS COMPOSED OF:
"         HEADER,GI88,GI84,GI82,CHEKOV,DCODE,DBLOCK
"       LINE IS THE INPUT BUFFER
"       LENGTH(I) CONTAINS THE NUMBER OF ENTRIES IN THE I-TH DIRECTORY
"       THE VALUE OF BATCH INDICATES THE ACCESS MODE:
"         BATCH=0    ALL STUDENTS
"         BATCH=1    ALL STUDENTS IN 1 CURRICULUM
"         BATCH=2    ALL STUDENTS OF 1 ADVISOR IN 1 CURRICULUM
"         BATCH=3    A SINGLE STUDENT
"       THE VALUE OF STATUS INDICATES THE PROGRESS OF A SEARCH:
"         STATUS=1    SEARCH JUST STARTING
"         STATUS=2    SEARCH IN PROGRESS
"         STATUS=3    LAST ACCESS OF SEARCH
"         STATUS=4    DONE WITH OLF
"       THE FOLLOWING VARIABLES ARE USED BY THE PROCESSORS:
"         SLAVE   IS THE IMAGE OF MASTER
"         VEC     IS THE IMAGE OF VECTOR
"         LL      IS THE IMAGE OF LENGTH
"         NODES   IS THE IMAGE OF LAST
"
      REAL*8 CURRIC/'CURRIC'/,VECTOR(260),LAST(4)/4*'DUMMY'/,%
        OUT,GIB8(5),FLAG/'????????'/,LOCATE/'LOCATE'/,PROC,%
        DONE/'DONE'/,ANS(4)/'CRRCLM','ADVISOR','STUDENT','PROCESSOR'/,%
        DUMMY/'DUMMY'/,PROCS(5)/'CRFILE','DLFILE','CALC',%
        'DUD','USERA'/,LIST(2)/'CRLIST','DLLIST'/,DCODE(12),%
        MASTER(4),SLAVE(4),VEC(260),PRCSSR/'PROCESSOR'/,%
        DUPLIC(4),NODES(4),EQUIV,DATA/'D A T A'/,%
```

```
        EQU(8),PROB(8)/3*' ','N A M E ','D   A B ','O V E   ',%
        2*' '/,NAME2
      INTEGER*2 HEADER(32),STAR/'*'/,$$/'$$'/,GIB2(20),%
        CHEKOV(12),DBLOCK(56,12),LINE(72),BL/' '/
      INTEGER ERR,OK,STATUS,BATCH/0/,AUX,LLEN(3),SKIP/0/,%
        L1(3)/1,201,241/,L3(3)/0,20,0/,TYPE(4)/2,5,1,5/,%
        PGM,ERROR,CARD/0/,DEBUG/0/,YES/'YES'/,NO/'NO'/,PR,%
        COPY(3)/3*0/,LENGTH(3)/3*1/,PART(5,5),%
        FINI,ROWS/5/,BEGIN/0/,DUP,LL(3),BB/0/,PTR,V1(2)/'INTO','FROM'/
      REAL GIB4(16),V(2)/' ','DEST'/
      COMMON HEADER,GIB8,GIB4,GIB2,CHEKOV,OCODE,DBLOCK,%
        VEC,SLAVE,NODES,LL
      EQUIVALENCE (HEADER(1),EQU(1)),(LINE(1),EQUIV)
      COMPLEX*16 END(2)/'NORMALLY','IN ERROR'/,%
        POSSIB(2)/'PROCESSOR','SEARCH'/
C
C       INITIALIZATION
C
C       LOAD PROCESSOR ACCESS RIGHTS TABLE (PART)
C
      CALL OPEN(1,'PART','INPUT')
      READ(1,2)((PART(I,J),J=1,5),I=1,ROWS)
2     FORMAT(5(I1,1X))
      CALL CLOSE(1)
      IF (DEBUG.EQ.1)WRITE(6,2000)((PART(I,J),J=1,5),I=1,ROWS)
2000  FORMAT(' *PART:'/(1X,5I2))
C
C       LOAD THE LEVEL-1 DIRECTORY CURRIC
C
      CALL LOAD(1,CURRIC,0,LAST,VECTOR,LENGTH,CURRIC,0)
C
C       DETERMINE INPUT MODE--CARDS OR TERMINAL
C
3     WRITE(6,4)
4     FORMAT(' INPUT ON CARDS--TYPE YES OR NO')
      READ(5,5) IJK
5     FORMAT(A4)
      IF (IJK.EQ.NO) GO TO 99
      IF (IJK.NE.YES) GO TO 3
      CARD=1
99    PR=1
C
C       REQUEST PROCESSOR (OR SEARCH) COMMAND
C
6     WRITE(6,7) POSSIB(PR)
7     FORMAT(' TYPE ',A9,' REQUEST'/)
      IF (SKIP.NE.0) GO TO 10
      READ(5,8) LINE
8     FORMAT(72A1)
      IF (CARD.NE.1) GO TO 12
10    WRITE(6,11) LINE
11    FORMAT(1X,72A1)
12    SKIP=0
      STATUS=1
20    ERR=0
      DUP=0
      OK=0
```

```
      FINI=0
      INCR=0
"
"      TEST CURRENT STATUS
"
      IF (STATUS-2) 30,200,1000
"
"      DETERMINE THE NEW SEARCH STRATEGY
"
30    ITYPE=5
      LEN=8
      ISTART=1
"
"      TRANSLATE 1ST FIELD
"
      CALL CONVRT(OUT,ITYPE,LINE,LEN,AUX,1,ERR,&555)
      IF (OUT.EQ.LOCATE) GO TO 45
      IF (OUT.NE.DONE) GO TO 40
"
"      FINISHED WITH THIS PROCESSOR
"
35    STATUS=4
      GO TO 150
"
"      INPUT CANNOT BE TRANSLATED
"
40    ERR=5
      GO TO 555
"
"      TRANSLATE 2ND FIELD
"
45    ISTART=11
      CALL CONVRT(OUT,ITYPE,LINE,LEN,AUX,11,ERR,&555)
      DO 50 I=1,4
"
"      DETERMINE BATCH ACCESS LEVEL FOR THE NEW SEARCH
"
      IF (OUT.EQ.ANS(I)) GO TO 52
50    CONTINUE
      GO TO 40
"
"      IS THIS A REQUEST FOR A NEW PROCESSOR?
"
52    IF (PR.EQ.1) GO TO 56
      IF (I.EQ.4) GO TO 58
      IF (I.LE.BATCH+1) GO TO 55
"
"      MUST SPECIFY A NODE NEARER TO THE ROOT
"
53    ERR=6
      GO TO 555
55    BATCH=I-1
      IF (LINE(21).NE.STAR) GO TO 60
"
"      THE LAST NODE SPECIFIED IS THE ROOT OF THE
"      SEARCH SUBTREE
"
```

```
56      MASTER(I)=FLAG
        LLEN(I)=0
        IF(I.EQ.3) GO TO 57
        I=I+1
        GO TO 56
57      STATUS=1
        GO TO 150
"
"        MUST TRANSLATE THIS NODE NAME
"
58      IF (LINE(21).EQ.STAR) GO TO 132
60      ISTART=21
        ITYPE=TYPE(I)
        IF (I.NE.3) GO TO 120
        IF (BATCH.EQ.2) LEN=9
"
"        TRANSLATE 3RD FIELD
"
120     CALL CONVRT(OUT,ITYPE,LINE,LEN,AUX,ISTART,ERR,&555)
        MASTER(I)=OUT
"
"        PROCESS THE NEW LINE
"
        IF (I.NE.4) GO TO 130
        DO 122 PGM=1,ROWS
        IF (OUT.EQ.PROCS(PGM)) GO TO 125
122     CONTINUE
        ERR=5
125     PROC=PROCS(PGM)
        IF (DEBUG.EQ.1) WRITE(6,126) PROC
126     FORMAT(' *THE ACTIVE PROCESSOR IS ',A8)
        RB=0
"
"        GET THE NEXT SEARCH SPECIFICATION
"
        PR=2
        BATCH=0
        GO TO 6
130     IF (PR.NE.1) GO TO 133
132     ERR=11
        GO TO 555
133     IF (I.NE.3) GO TO 135
        BATCH=3
        STATUS=1
        GO TO 150
135     IF (I.EQ.1) CALL RENAME($$,MASTER(1))
        IF (I.EQ.2) MASTER(4)=MASTER(2)
        BATCH=BATCH+1
"
"        READ NEXT LINE OF THE SEARCH SPECIFICATION
"
        GO TO 6
"
"        ACCESS A STUDENT FILE
"
150     IF (STATUS.NE.4) GO TO 200
"
```

```
"       PREPARING TO TERMINATE JOB
"
152     WRITE(6,155)
155     FORMAT(' GOING INTO FINAL WRAPUP')
        IF (COPY(3).EQ.1) CALL LOAD(3,DUMMY,COPY(3),LAST,VECTOR,%
          LENGTH,DUMMY,0)
        WRITE(6,165)
165     FORMAT(' AFTER THIS JOB ENDS, TYPE:',' RUNN COPY')
        STOP
"
"       ACCESSING STUDENT FILES
"
200     IF (DEBUG.EQ.1) WRITE(6,2030) STATUS,BATCH,(MASTER(MM),MM=1,3)
2030    FORMAT(' *SEARCH STRATEGY DETERMINED:'%
          ' STATUS=',I1,5X,'BATCH=',I1,/' MASTER=',3A12)
        ILK=0
        KK=BATCH+1
        I=1
        IF (STATUS.GT.1) GO TO 215
        STATUS=2
        BB=0
        KM=1
210     IF (MASTER(I).NE.FLAG) GO TO 220
"
"       FIND THE NAME OF THE NEXT LEVEL-I DIRECTORY
"
215     I=KM
        MASTER(I)=VECTOR(L1(I)+LLEN(I)+L3(I))
        LLEN(I)=LLEN(I)+1
217     DUP=1
        GO TO 222
"
"       IF MASTER(I) IS THE CURRENT LEVEL-I DIRECTORY,
"       IT MUST EXIST
"
220     IF (MASTER(I).EQ.LAST(I+1)) GO TO 217
"
"       SEARCH FOR MASTER(I) IN DIRECTORY(I)
"
        CALL BISECT(VECTOR,LENGTH,MASTER(I),DUP,PTR,I,LAST)
        LLEN(I)=PTR-L1(I)+1
        IF (DUP.EQ.0) GO TO 222
        IF (I.EQ.2) MASTER(2)=VECTOR(PTR+L3(2))
222     DUPLIC(I)=MASTER(I)
        IF (I.EQ.2) DUPLIC(4)=MASTER(4)
        IF (DUP.NE.0) GO TO 225
        IF (I.EQ.3) GO TO 260
"
"       THE DIRECTORY JUST REQUESTED IS UNKNOWN
"
        GO TO 300
225     NAME2=VECTOR(L1(I)+LLEN(I)-1)
        IF (I.EQ.3) GO TO 260
"
"       LOAD FILE NAMED MASTER(I) INTO LEVEL-(I+1) DIRECTORY
"
        KARD=CARD
```

```
      CALL LOAD(I+1,MASTER(I),COPY(I+1),LAST,VECTOR,LENGTH,%
        NAME2,KARD)
      I=I+1
      KM=KM+1
      IF (LENGTH(I).NE.0) GO TO 210
      IF (BATCH.NE.3) GO TO 240
      DUP=0
      STATUS=3
      DUPLIC(3)=MASTER(3)
      PTR=L1(3)
      GO TO 290
240   LLEN(I)=LLEN(I)+1
      ILK=1
      I=I-1
      GO TO 261
260   OUT=MASTER(3)
      IF (BATCH.EQ.3) GO TO 263
"
"       IS THE BATCH ACCESS COMPLETED WITH THIS LAST FILE?
"
261   J=3
262   IF (LLEN(J).LT.LENGTH(J)) GO TO 265
      J=J-1
      IF (J.GE.KK) GO TO 262
263   STATUS=3
      GO TO 290
265   KM=J
      IF (J.NE.3) GO TO 267
      MASTER(3)=FLAG
      GO TO 290
267   KI=LLEN(J)
      DO 270 L=J,3
      LLEN(L)=0
270   MASTER(L)=FLAG
      LLEN(J)=KI
290   IF (ILK.EQ.0) GO TO 295
"
"       AN EMPTY DIRECTORY WAS SPECIFIED
"
      ERR=12
      GO TO 555
"
"       IF THE FILE WAS FOUND--GO PROCESS IT
"
295   IF (DUP.EQ.1) GO TO 350
      IF (I.NE.3) GO TO 300
"
"       DOES THE CURRENT PROCESSOR HAVE DIRECT ACCESS OK?
"
      IF (PART(PGM,4).EQ.1) GO TO 305
"
"       THE LEVEL-J FILE JUST REQUESTED IS MISSING
"
300   ERR=6+I
      FINI=1
      GO TO 555
"
```

```
"       TRY TO ACCESS THE STUDENT FILE DIRECTLY
"
305     CALL OPEN(1,OUT,'INPUT',K)
        IF (K.EQ.0) GO TO 320
"
"       FILE DOES NOT EXIST
"
        DO 310 I=1,8
310     EQU(I)=PROB(I)
        DO 315 I=1,9
315     HEADER(I)=LINE(I+20)
        GO TO 500
"
"       THE FILE EXISTS--FIND OUT ITS OWNER
"
320     ERR=3
        READ(1,355) HEADER,GI88
        CALL CLOSE(1)
        GO TO 555
350     IF (DEBUG.EQ.1) WRITE(6,352) OUT
352     FORMAT(' *NOW READING STUDENT FILE ',A8)
"
"       READ IN THE APPROPRIATE AMOUNT OF THE STUDENT FILE
"
        CALL OPEN(1,OUT,'INPUT')
        READ(1,355) HEADER,GI88,GI84,GI82,CHEKOV
355     FORMAT(32A1,5A8/16A4/20A2,12A1)
        IF (PART(PGM,1).EQ.1) READ(1,360) DCODE,DBLOCK
360     FORMAT(6A8/6A8/(28A2))
        CALL CLOSE(1)
"
"       MAKE SURE THAT THIS STUDENT IS IN THE RIGHT CURRICULUM
"
        IF (DUPLIC(1).EQ.GI88(2)) GO TO 500
"
"       WRONG CURRICULUM--ADVISOR COUNSELS >1 CURRICULUM
"
        DUP=0
        ERR=4
        GO TO 555
"
"       LOCK FOR INPUT IF IN INDIVIDUAL ACCESS MODE
"       (BATCH=3) OR IF ON 1ST ACCESS OF A BATCH ACCESS
"
500     IF (BATCH.EQ.3) GO TO 501
        IF (BB.EQ.0) GO TO 501
        GO TO 555
"
"       BUFFER INPUT TO THE PROCESSOR
"
501     BB=1
        CALL OPEN(12,'BUFFER','OUTPUT')
        WRITE(12,8) LINE
"
"       REQUEST DATA FOR PROCESSOR (IF ANY)
"
        NUM=1
```

```
502   ITRY=1
503   WRITE(6,504)
504   FORMAT(' TYPE DATA (IF ANY)'/)
      READ(5,8) LINE
      IF (BATCH.EQ.3) GO TO 508
      IF (EQUIV.EQ.DATA) GO TO 530
      GO TO 513
508   DO 510 I=1,9
      IF (HEADER(I).NE.LINE(I)) GO TO 513
510   CONTINUE
      GO TO 530
513   IF (CARD.NE.1) GO TO 514
      SKIP=1
      GO TO 540
"
"      ACCEPT A CARRIAGE RESTORE AS END OF DATA
"      WHEN IN TERMINAL INPUT MODE
"
514   DO 515 I=1,72
      IF (LINE(I).NE.BL) GO TO 520
515   CONTINUE
      GO TO 540
"
"      CANNOT TRANSLATE LAST LINE
"
"
"      GET FOUR CHANCES TO GET IT RIGHT WHEN
"      IN TERMINAL INPUT MODE
"
520   IF (ITRY.GT.4) GO TO 525
      IF (BATCH.EQ.3) WRITE(6,522)(LINE(M),M=1,9)
522   FORMAT(1X,9A1,' DOES NOT MATCH THE SSN OF THE ',%
        'REQUESTED FILE')
      IF (BATCH.LT.3) WRITE(6,523)
523   FORMAT(' TYPE "DATA"  " IN COLUMNS ',%
        '1 TO 8 FOR BATCH MODE DATA INPUT')
      ITRY=ITRY+1
      GO TO 503
525   ERR=5
      GO TO 540
"
"      TRANSFER THIS LINE TO THE BUFFER
"
530   WRITE(12,8) LINE
      IF (CARD.EQ.1) WRITE(6,11) LINE
      NUM=NUM+1
      IF (NUM.LE.4) GO TO 503
"
"      TOO MUCH INPUT
"
      ERR=2
540   CALL CLOSE(12)
"
"      PRINT STATUS BEFORE CALLING PROCESSOR
"
555   CALL OUTPUT(ERR,PROC,BATCH,DUPLIC,LAST,I,GIB8,%
        LINE,HEADER,ISTART,LEN,NAME2,OK)
```

```
C
C       IF THERE'S BEEN AN ERROR ABOVE, SKIP PROCESSONG STEP
C
      IF (1-OK) 556,560,637
C
C       SERIOUS ERRORS--CAUSE TERMINATION IN CARD MODE
C
556   IF (CARD.EQ.0) GO TO 99
      WRITE(6,557)
557   FORMAT(' SINCE INPUT IS ON CARDS, PROGRAM MUST STOP')
      GO TO 152
C
C       PREPARE FOR BRANCH TO CURRENT PROCESSOR
C
560   ERROR=0
      DO 562 I=1,4
562   SLAVE(I)=DUPLIC(I)
      DO 568 I=1,3
      K=LENGTH(I)
      LL(I)=K
      IF (I.EQ.2) K=K+L3(I)
      M=L1(I)
      DO 567 J=M,K
567   VEC(J)=VECTOR(J)
568   NODES(I)=LAST(I)
C
C       BRANCH TO PROCESSOR
C
569   GO TO (570,575,580,585,590),PGM
570   CALL CRFILE(ERROR,INCR,DUP)
      GO TO 600
575   CALL DLFILE(ERROR,INCR)
      GO TO 600
580   CALL CALC(GPA,HOURS,ERROR,NUM,1,0)
      GO TO 600
585   CALL DUD
      GO TO 600
590   CALL USERA(ERROR,FINI)
C
C       IF THE PROCESSOR SIGNALLED AN ERROR, DISALLOW OUTPUT
C
600   IF (ERROR.NE.0) GO TO 630
C
C       IS THERE DIRECTORY EDITING AUTHORITY?
C       (EDITING NOT ALLOWED AFTER STATISTICS ACCESS)
C
      IF (FINI.EQ.1) GO TO 630
      IF (PART(PGM,3).EQ.0) GO TO 610
C
C       DOES IT WANT TO EDIT THE DIRECTORY?
C
      IF (INCR.EQ.0) GO TO 610
      NAME2=MASTER(4)
      CALL REVISE(VECTOR,LENGTH,OUT,OUT,PTR,3,INCR,LAST,ERROR,NAME2)
C
C       IF THE EDITING SUBPROGRAM REPORTS AN ERROR, DISALLOW EDITING
C
```

```
      IF (ERROR.NE.0) GO TO 630
"
"       THE DIRECTORY WILL BE REWRITTEN ONLY IF
"       THIS FLAG IS SET
"
      COPY(3)=1
      K=INCR
      IF (K.EQ.-1) K=2
      CALL OPEN(1,LIST(K),'MOD')
      WRITE(1,605) V(K),OUT,V1(K),DUPLIC(2)
605   FORMAT(A4,1X,A8,10X,A4,1X,A8)
      CALL CLOSE(1)
"
"       IS OUTPUT OF THE STUDENT FILE AUTHORIZED?
"
610   IF (PART(PGM,2).EQ.0) GO TO 630
      CALL OPEN(1,OUT,'OUTPUT')
      WRITE(1,615) HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK
615   FORMAT(32A1,5A8/16A4/20A2,12A1/6A8/6A8/(28A2))
      CALL CLOSE(1)
"
"       WRITE COMPLETION MESSAGE
"
630   ERROR=ERROR+1
      WRITE(6,635) PROC,END(ERROR)
635   FORMAT(1X,A8,' ENDED ',A16)
"
"       AFTER STATISTICS WRAPUP ASK FOR NEW PROCESSOR
"
637   IF (FINI.EQ.1) GO TO 6
"
"       CHECK SEARCH STATUS
"
      GO TO 20
"
"       PREPARE FOR STATISTICS WRAPUP--IF ANY
"
1000  IF (PART(PGM,5).EQ.0) GO TO 6
      ERR=1
      FINI=1
      GO TO 555
      END
"
"
"
"
"       OLF PROCESSORS
"
"       PROCESSOR CRFILE CREATES EACH NEW STUDENT FILE
"
      SUBROUTINE CRFILE(ERROR,INCR,DUP)
      REAL*8 DCODE(12),GIB8(5),SLAVE(4),OUT,BLANK/'  '/,VEC(260),NODES(4)
      INTEGER*2 LLINE(72),DBLOCK(56,12),GIB2(20),CHEKOV(12),HEADER(32),&
        M(20),BL/' '/,ZERO/'0'/
      REAL GIB4(16)
      INTEGER ERROR,DEBUG/0/,DUP,ENTRY,LL(3),$$*2/'$$'/
      COMMON HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK,&
```

```
        VEC,SLAVE,NODES,LL
      EQUIVALENCE (GIB8(1),M(1))
      IF (DEBUG.EQ.1) WRITE(6,5)
5     FORMAT(' *ENTERING CRFILE')
      CALL OPEN(12,'BUFFER','INPUT')
      READ(12,8) LLINE
8     FORMAT(72A1)
      IF (DUP.EQ.0) GO TO 20
C
C      THE PROPOSED FILE ALREADY EXISTS
C
      WRITE(6,10)
10    FORMAT(' THE PROPOSED FILE ALREADY EXISTS')
      GO TO 990
C
C      DOES THE NAME FIELD CONTAIN ANY TYPOS?
C
20    CALL CONVRT(OUT,3,LLINE,20,JUNK,31,ERROR,&990)
C
C      IS THE PROPOSED ENTRY DATE OK?
C
      CALL TIMES(ENTRY,LLINE,51,61,K,ERROR,&990)
      IF (K.GE.0) GO TO 60
      WRITE(6,40)(LLINE(I),I=51,65)
40    FORMAT(1X,15A1,' IS PAST--CAN''T BE AN ENTRY DATE')
      GO TO 990
C
C      INITIALIZE THE NEW FILE
C
60    DO 70 I=1,30
70    HEADER(I)=LLINE(I+20)
      HEADER(10)=BL
      HEADER(31)=BL
      HEADER(32)=BL
      DO 90 I=1,12
90    CHEKOV(I)=ZERO
      DO 100 I=1,16
100   GIB4(I)=0.0
      DO 110 I=1,20
110   GIB2(I)=0
      GIB2(2)=ENTRY
      GIB8(1)=BLANK
      GIB8(2)=SLAVE(1)
      GIB8(3)=SLAVE(4)
      GIB8(4)=BLANK
      GIB8(5)=BLANK
C
C      INSERT COLLEGE CODE FROM 2ND BYTE OF CURRICULUM CODE
C
      M(1)=M(6)
C
C      INCR=1 TELLS OLF TO ADD THE NEW FILE TO THE ADVISOR LIST AND
C      TO THE LIST OF NEW FILES
C
      INCR=1
      K=0
C
```

```
"       IF THERE IS NO MORE DATA--RETURN
"
        READ(12,8,END=1000) LLINE
        K=1
"
"       PROCESS CO-CURRICULUM FIELD
"
        CALL CONVRT(OUT,2,LLINE,8,JUNK,21,ERROR,&115)
        GO TO 300
"
"       LET MISTAKE IN CO-CURRICULUM GO BY WITH A WARNING
"
115     ERROR=0
        WRITE(6,120)
120     FORMAT(' CREATING THE FILE WITHOUT CO-CURRICULUM')
        GO TO 1000
"
"       MAKE SURE THAT THE PROPOSED CO-CURRICULUM EXISTS
"
300     CALL RENAME($$,OUT)
        CALL BISECT(VEC,LL,OUT,DUP,PTR,1,NODES)
        IF (DUP.EQ.1) GO TO 350
        WRITE(6,310) OUT
310     FORMAT(' CURRICULUM ',A8,' IS UNKNOWN')
        GO TO 115
350     GIB8(5)=OUT
"
"       GET CO-COLLEGE CODE FROM CO-CURRICULUM CODE
"
        M(13)=M(18)
        GO TO 1000
990     ERROR=1
1000    IF (DEBUG.EQ.1) WRITE(6,1005) ERROR
1005    FORMAT(' *LEAVING CRFILE--ERROR=',I1)
        RETURN
        END
"
"
"
"
"       PROCESSOR DLFILE DELETES EACH OBSOLETE STUDENT FILE
"
        SUBROUTINE DLFILE(ERROR,INCR)
        REAL*8 VEC(260),DCODE(12),GIB8(5),SLAVE(4),OUT,NODES(4),GIB4*4(16)
        INTEGER*2 LLINE(72),DBLOCK(56,12),GIB2(20),CHEKOV(12),HEADER(32)
        INTEGER ERROR,DEBUG/0/,LL(3)
        COMMON HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK,%
          VEC,SLAVE,NODES,LL
        IF (DEBUG.EQ.1) WRITE(6,5)
5       FORMAT(' *ENTERING DLFILE')
"
"       COMPARE HEADER TO THE NAME FIELD IN THE DELETE REQUEST
"
20      CALL OPEN(12,'BUFFER','INPUT')
        READ(12,22) LLINE
22      FORMAT(72A1)
        CALL CLOSE(12)
```

```
      DO 25 I=11,30
      IF (LLINE(I+20).NE.HEADER(I)) GO TO 30
25    CONTINUE
      GO TO 40
C
C      NAMES DON'T MATCH
C
30    WRITE(6,35)(HEADER(I),I=11,30)
35    FORMAT(' NAMES DIFFERENT--FILE FOR ',20A1)
      GO TO 990
C
C      ADD COMPLETION DATE TO THE FILE
C
40    CALL OPEN(1,'DATES','INPUT')
      READ(1,45) GIB2(7)
45    FORMAT(I5)
      CALL CLOSE(1)
C
C      INCR=-1 TELLS OLF TO DELETE THE FILE FROM THE ADVISOR
C      LIST AND TO ADD IT TO THE OBSOLETE FILE LIST
C
      INCR=-1
      GO TO 1000
990   ERROR=1
1000  IF (DEBUG.EQ.1) WRITE(6,1005) ERROR
1005  FORMAT(' *LEAVING DLFILE--ERROR=',I1)
      RETURN
      END
C
C
C
C
C
C      PROCESSOR CALC CALCULATES GPA'S FROM VARIOUS SETS OF GRADES:
C      GRADES RECORDED IN THE FILE BUT NOT INCLUDED IN GPA, OR
C      THE UNION OF UP TO ANY 4 OF THE FOLLOWING:
C      1) ALL COURSES WITHIN THE MAJOR AREA OF STUDY
C      2) ALL COURSES WITHIN THE MINOR AREA(S) OF STUDY
C      3) ALL COURSES WITHIN THE MAJOR AND MINOR AREAS OF STUDY
C      4) ALL COURSES WITHIN A SPECIFIC DEPARTMENT
C      5) ALL COURSES PAST A GIVEN SEMESTER
C      6) ALL COURSES PAST A GIVEN ACADEMIC STATUS
C      7) ALL COURSES WITHIN A GIVEN NUMBER OF HOURS
C      8) ALL COURSES INCLUDING AND ABOVE A GIVEN LEVEL (E.G., 100)
C      THE DEFAULT VALUES ARE:
C        ALL DEPARTMENTS,
C        MAJOR, MINOR, AND GENERAL EDUCATION COURSES,
C        100 LEVEL,
C        FROM FRESHMAN STATUS,
C        FROM ENTRY, AND
C        ALL HOURS COMPLETED
C      GRADES USED IN CALCULATIONS ARE A,B,C,D,E,AB.
C      AB IS REPRESENTED INTERNALLY AS 5.
C      GRADES NOT YET INCLUDED IN A GPA ARE FLAGGED WITH
C      AN ASTERISK (E.G., A*).  AN 'IP' GRADE CODE STANDS
C      FOR 'IN PROGRESS'.
C
```

```
      SUBROUTINE CALC(GPA,HOURS,ERROR,NUM,PRINT,SET)
      INTEGER*2 LLINE(72),CBLOCK(56,12),GIB2(20),HEADER(32),CHEKOV(12),%
        GRADE,IJK,AB/'AB'/,G(6)/'G','E','D','C','B','A'/,%
        IP/'IP'/
      INTEGER ERROR,SUM1,SUM2,SUM3,SUM4,DEPT(12),DDATE,AUX,%
        FINI,S,T,DEBUG/0/,CUTOFF,END,PRINT,SET,H,HR,LONG(6)/4*8,2*3/,%
        L1(2)/2,1/,L2(2)/2,1/,LL(3),TYPE(6)/4*5,2*6/,PP,KOL/1/
      REAL*8 VEC(260),OUT,SLAVE(4),DCODE(12),GIB8(5),A(3)/'MAJOR',%
        'MINOR',' '/,B(4)/'FRESHMAN','SOPHOMORE','JUNIOR','SENIOR'/,%
        KEY(6)/'TYPE','DEPT','STATUS','SEMESTER','HOURS','LEVEL'/,%
        NODES(4),GIB4*4(16)
      COMMON HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK,%
        VEC,SLAVE,NODES,LL
      IF (DEBUG.EQ.1) WRITE(6,5) SET
5     FORMAT(' *ENTERING CALC--SET=',I1)
C
C      INITIALIZATIONS
C
      PP=PRINT
      NUMBER=0
      DO 7 J=1,12
7     DEPT(J)=0
      MM=0
      IFAIL=0
      NUM=0
      HOURS=0.0
      GPA=0.0
      NMAJOR=0
      T=0
      SUM1=0
      SUM2=0
      SUM3=0
      SUM4=0
      KK=1
      S=0
C
C      IF SET = 1, THEN LEVEL, DDATE, ETC. FROM THE LAST
C      USE OF THE PROCESSOR ARE TO BE USED AGAIN
C
      IF (SET.NE.0) GO TO 1900
      I=0
C
C      THERE ARE IEND DEPT. BLOCKS USED IN THIS FILE
C
      IEND=GIB2(10)
      DDATE=0
      ITOP=3
      IBOT=0
      LEVEL=100
C
C      FIND CURRECT SEMESTER
C
      CALL OPEN(1,'DATES','INPUT')
      READ(1,10) NOW
10    FORMAT(I5)
      CALL CLOSE(1)
C
```

```
C       READ SEARCH CLASS DESCRIPTION(S)
C
        CALL OPEN(12,'BUFFER','INPUT')
C
C       FIRST LINE IN BUFFER IS AN OLF COMMAND,
C       NOT A DATA LINE FOR THIS PROCESSOR
C
        READ(12,20) LLINE
17      READ(12,20,END=1000) LLINE
20      FORMAT(72A1)
        I=1
C
C       DETERMINE TYPE OF DESCRIPTION
C
        CALL CONVRT(OUT,5,LLINE,8,AUX,11,ERROR,&2990)
        DO 30 J=1,6
        IF (OUT.EQ.KEY(J)) GO TO 40
30      CONTINUE
        GO TO 2800
40      IF (J.EQ.4) GO TO 400
        MM=1
C
C       CONVERT THE DATA FIELD
C
50      CALL CONVRT(OUT,TYPE(J),LLINE,LONG(J),AUX,11+10*MM,%
          ERROR,&2990)
C
C       SET THE STRATEGY
C
        GO TO (100,200,300,400,500,600),J
C
C       MAJOR AND/OR MINOR COURSES
C
100     DO 115 K=1,3
        IF (A(K).EQ.OUT) GO TO 120
115     CONTINUE
        GO TO 2800
120     IF (OUT.EQ.A(3)) GO TO 170
        IF (L1(K).LT.ITOP) ITOP=L1(K)
        IF (L2(K).GT.IBOT) IBOT=L2(K)
        KK=KK+1
        MM=2
        IF (KK.LE.2) GO TO 50
        GO TO 17
170     IF (KK.EQ.1) GO TO 2800
        GO TO 17
C
C       COURSES IN A SPECIFIC DEPARTMENT
C
200     DO 250 KOL=1,IEND
        IF (OUT.EQ.DCODE(KOL)) GO TO 270
250     CONTINUE
        WRITE(6,260) OUT
260     FORMAT(' "',A8,'" IS NOT A DEPT. USED IN THIS FILE')
        GO TO 3000
270     I=KOL
        IEND=KOL
```

```
      GO TO 17
"
"       COURSES AFTER REACHING A SPECIFIC ACADEMIC STATUS
"
300   DO 350 K=1,4
      IF (OUT.EQ.B(K)) GO TO 370
350   CONTINUE
      GO TO 2800
370   K=GIB2(K+2)
      IF (K.EQ.0) K=NOW+10
      IF (K.GT.DDATE) DDATE=K
      GO TO 17
"
"       COURSES TAKEN DURING OR AFTER A GIVEN SEMESTER
"
400   CALL TIMES(K,LLINE,21,31,KR,ERROR,&3000)
      IF (K.GT.DDATE) DDATE=K
      GO TO 17
"
"       COURSES WITHIN THE LAST SO-MANY HOURS
"       GIB2(15) IS THE NUMBER OF COURSES COMPLETED
"
500   IF (GIB2(15).EQ.0) GO TO 3000
      FINI=0
      H=0
      HR=AUX*2
"
"       FIND LAST HR HOURS WORTH OF CREDIT BY SEARCHING
"       BACKWARDS IN TIME THROUGH THE FILE.
"       INITIALIZE CUTOFF DATE TO PRESENT.
"       SET END TO STUDENT'S ENTRY DATE
"
      CUTOFF=NOW
      END=GIB2(2)
510   DO 525 J=1,IEND
      K2=(DBLOCK(3,J)-1)*5+7
      DO 525 K1=7,K2,5
"
"       REJECT THIS COURSE IF NOT TAKEN IN SEMESTER AT DATE CUTOFF
"       REJECT THIS COURSE IF NOW IN PROGRESS
"       REJECT THIS COURSE IF NOT YET INCLUDED IN GPA
"
      IF (DBLOCK(K1+1,J).NE.CUTOFF) GO TO 520
      GRADE=DBLOCK(K1+3,J)
      IF (GRADE.EQ.IP) GO TO 520
"
"       LOOK FOR * IN RIGHT BYTE
"
      IJK=GRADE-(GRADE/256-255*256-1)*256
      IF (IJK.EQ.92) GO TO 520
      H=H+DBLOCK(K1+2,J)
"
"       GONE BACK FAR ENOUGH IF H.GE.HR OR
"       IF HAVE BACK-TRACKED THROUGH TO ENTRY DATE
"       (I.E., CUTOFF.EQ.END)
"
520   IF (CUTOFF.LE.END) GO TO 522
```

```
      IF (H.LT.HR) GO TO 525
522   FINI=1
525   CONTINUE
      IF (FINI.EQ.0) GO TO 540
      IF (CUTOFF.GT.DDATE) DDATE=CUTOFF
      GO TO 17
540   CUTOFF=CUTOFF-1
      IF (MOD(CUTOFF,10).EQ.0) CUTOFF=CUTOFF-7
      GO TO 510
"
"       ADVANCED LEVEL COURSES
"
600   IF (AUX.LE.0) GO TO 2800
      LEVEL=AUX
      GO TO 17
"
"       INCLUDE NEW GRADES IN GPA
"
1000  IF (I.NE.0) GO TO 1900
      I=1
      T=1
      IADV=0
      GO TO 1900
"
"       UPDATE GIB4, GIB2
"
1050  GIB4(2)=GIB4(2)+HOURS
      GIB4(1)=GIB4(1)+HOURS-IFAIL*.5
      GIB4(3)=GIB4(3)+IADV*.5
      GIB4(8)=GIB4(8)+SUM1*.5
      IF (GIB4(2).NE.J.0) GIB4(6)=GIB4(8)/GIB4(2)
      GIB4(10)=GIB4(10)+SUM3*.5
      GIB4(9)=GIB4(9)+SUM4*.5
      IF (GIB4(9).NE.0.0) GIB4(7)=GIB4(10)/GIB4(9)
      GIB2(15)=GIB2(15)+NUM
      GIB2(14)=GIB2(14)+NMAJOR
      GIB2(13)=GIB2(13)+NUM
      PP=0
      T=2
"
"       UPDATE DEPARTMENT INFORMATION BLOCKS
"
      I=1
1070  SUM1=0
      SUM2=0
      NUMBER=0
      IFAIL=0
      IF (DEPT(I).EQ.1) GO TO 2050
      GO TO 1200
1080  DBLOCK(1,I)=SUM2-IFAIL
      DBLOCK(2,I)=DBLOCK(1,I)
      DBLOCK(5,I)=SUM1
1200  I=I+1
      IF (I.LE.IEND) GO TO 1070
      GO TO 3000
"
"       FIND GPA OF SEARCH CLASS
```

```
"       ITCP=3 -> NOT RESTRICTED TO MAJOR AND/OR MINOR
"
1900    IF (DEBUG.EQ.1) WRITE(6,1910) LEVEL,DDATE,ITOP,IBOT,I,IEND
1910    FORMAT(' LIMITS: LEVEL=',I3,'   DDATE=',I5,%
          '   ITOP=',I1,'   IBOT=',I1,'   DEPTS ',I2,' TO ',I2)
2000    IF (ITOP.EQ.3) GO TO 2050
"
"       RIGHT TYPE?
"
        S=DBLOCK(4,I)
        IF (S.LT.ITOP) GO TO 2700
        IF (S.GT.IBOT) GO TO 2700
2050    IF (DEBUG.EQ.1) WRITE(6,2051) DCODE(I),I,%
          (DBLOCK(K3,I),K3=1,6)
2051    FORMAT(' NEW DEPT.=',A8,'DBLOCK(1->6,',I2,')=',%
          6I4)
        K2=(DBLOCK(3,I)-1)*5+7
        DO 2500 K1=7,K2,5
        K3=K1+4
        IF (DEBUG.EQ.1) WRITE(6,2055) (DBLOCK(K4,I),K4=K1,K3)
2055    FORMAT(' COURSE=',I3,'  DATE=',I5,'  HOURS*2=',I2,%
          '  GRADE=',A2,'  CODE=',I3)
"
"       WAS THE COURSE TAKEN TOO LONG AGO?
"
        IF (DBLOCK(K1+1,I).LT.DDATE) GO TO 2500
"
"       IS THE COURSE TOO LOW-LEVEL?
"
        LVL=DBLOCK(K1,I)
        IF (LVL.LT.LEVEL) GO TO 2500
        GRADE=DBLOCK(K1+3,I)
        DO 2100 J=1,6
"
"       HAS THIS COURSE BEEN INCLUDED IN A GPA YET?
"       (IS IT A,B,C,D,E,OR AB?)
"
        IF (T.EQ.1) GO TO 2080
2060    IF (GRADE.NE.G(J)) GO TO 2100
"
"       THIS COURSE BELONGS IN THE DESIRED GPA
"
        H=DBLOCK(K1+2,I)
        SUM1=SUM1+(J-1)*H
        SUM2=SUM2+H
        IF (J.LT.3) IFAIL=IFAIL+H
        NUMBER=NUMBER+1
        IJK=G(J)
        IF (J.EQ.1) IJK=AB
        IF (PP.EQ.1) WRITE(6,2063) DCODE(I),DBLOCK(K1,I),IJK
2063    FORMAT(' USING ',A6,I3,' WITH GRADE ',A2)
        IF (T.NE.1) GO TO 2500
"
"       THESE ARE JUST-RECEIVED GRADES--CHANGE "*" TO " "
"
        DBLOCK(K1+3,I)=G(J)
        DEPT(I)=1
```

```
      IF (LVL.GE.200) IADV=IADV+H
C
C       UPDATE HOURS AND GRADES IN MAJOR, IF APPROPRIATE
C
      IF (DBLOCK(4,I).NE.2) GO TO 2500
      SUM3=SUM3+(J-1)*H
      SUM4=SUM4+H
      NMAJOR=NMAJOR+1
      GO TO 2500
2080  GRADE=GRADE-28
      GO TO 2060
2100  IF (GRADE.LT.DBLOCK(K1+3,I)) GRADE=DBLOCK(K1+3,I)
2500  CONTINUE
      IF (T.EQ.2) GO TO 1080
2700  I=I+1
      IF (I.LE.IEND) GO TO 2000
      I=I-1
      IF (SUM2.EQ.0) GO TO 2520
      GPA=SUM1/FLOAT(SUM2)
      HOURS=SUM2*.5
2520  NUM=NUMBER
      IF (T-1) 3000,1050,1080
2800  K=MM+2
      WRITE(6,2810) OUT,K
2810  FORMAT(' "',A8,'" IS AN IMPROPER RESPONSE FOR FIELD ',I1)
2990  ERROR=1
3000  IF (PRINT.EQ.1) WRITE(6,3005) GPA,HOURS,NUM
3005  FORMAT(' LEAVING CALC   GPA=',F5.3,'  HOURS=',F6.2,%
        '  NUM=',I3)
      CALL CLOSE(12)
      RETURN
      END
C
C
C
C
C       PROCESSOR USERA COMPARES THE GPAS OF TWO SETS OF STUDENTS--
C       THOSE WHO TOOK A SPECIFIED COURSE BEFORE TAKING OTHER COURSES
C       IN A GIVEN DEPARTMENT, AND THOSE WHO DID NOT TAKE IT
C
      SUBROUTINE USERA(ERROR,FINI)
      INTEGER*2 LLINE(72),HEADER(32),CHEKOV(12),GIB2(20),%
        DBLOCK(56,12)
      INTEGER ERROR,COL/-1/,DEBUG/1/,SET,START/0/,N(2)/2*0/,%
        LL(3),FINI
      REAL*8 FILE(2)/'FILE1','FILE2'/,GIB8(5),VEC(260),SLAVE(4),%
        NODES(4),VERB(2)/'WITH','WITHOUT'/,DCODE(12),DEPT1,DEPT2
      REAL GIB4(16),G(2)/2*0./,H(2)/2*0./,SDEV(2)/2*0./
      COMMON HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK,%
        VEC,SLAVE,NODES,LL
      IF (DEBUG.EQ.1) WRITE(6,5) FINI
5     FORMAT(' *ENTERING USERA--FINI=',I1)
C
C       IF FINI = 1 THEN THIS IS THE STATISTICS WRAPUP CALL
C       GO CALCULATE THE MEAN AND THE STANDARD DEVIATION HOURS*GRADES
C
      IF (FINI.EQ.1) GO TO 1000
```

```
      IF (START.NE.0) GO TO 20
C
C      READ THE NAMES OF THE COURSE AND DEPARTMENT IN QUESTION
C
      CALL OPEN(12,'BUFFER','INPUT')
      READ(12,6)LLINE
6     FORMAT(72A1)
      CALL CONVRT(DEPT1,5,LLINE,8,JUNK,31,ERROR,&2150)
      CALL CONVRT(DEPT2,4,LLINE,8,NUM,41,ERROR,&2150)
C
C      THE REMAINING BUFFERED INFORMATION IS FOR PROCESSOR CALC
C      ADDING A CALC-FORM COMMAND FOR DEPT. DEPT1
C      CALC IGNORES THE FIRST LINE IN THE BUFFER
C
      CALL CLOSE(12)
      CALL OPEN(12,'BUFFER','MOD')
      WRITE(12,7) DEPT1
7     FORMAT(10X,'DEPT',6X,A8)
C
C      CREATE FILE1 & FILE2 TO CLEAR THEM OUT
C      GPA*HOURS FOR STUDENTS WITHOUT TEST COURSE GO IN FILE1
C      GPA*HOURS FOR STUDENTS WITH TEST COURSE GO IN FILE2
C
      DO 15 I=1,2
      CALL OPEN(I,FILE(I),'OUTPUT')
15    CALL CLOSE(I)
      START=1
20    J=1
C
C      HERE, K IS THE NUMBER OF DEPARTMENT BLOCKS USED
C      BY THE CURRENT STUDENT
C
      K=GIB2(10)
      IF (K.EQ.0) GO TO 3000
      DO 30 I=1,K
      IF (DCODE(I).EQ.DEPT2) GO TO 35
30    CONTINUE
C
C      HAS TAKEN NO COURSES IN THE SAME DEPARTMENT AS THE TEST COURSE
C
      GO TO 100
35    KK=DBLOCK(3,I)
      DO 37 M=1,KK
      IF (DBLOCK(2+M*5,I).EQ.NUM) GO TO 40
37    CONTINUE
C
C      HAS NOT TAKEN THE TEST COURSE
C
      GO TO 100
C
C      HAS TAKEN THE TEST COURSE
C
40    J=2
C
C      IF THE STUDENT HAS THE GIVEN DEPARTMENT IN THE SAME COLUMN AS THE
C      LAST STDUENT FILE SEEN BY CALC, THEN THE SEARCH STRATEGY USED BY
C      CALC DOES NOT CHANGE--SET=1 TELLS CALC TO LEAVE THE BUFFER
```

```
C       CLOSED AND REUSE ITS OLD SEARCH STRATEGY
C
100   SET=1
      DO 130 I=1,K
      IF (DCODE(I).EQ.DEPT1) GO TO 150
130   CONTINUE
C
C       STUDENT HAS NOT TAKEN ANY COURSES IN THE GIVEN DEPARTMENT
C
      GO TO 3000
C
C       IF I=COL THEN LEAVE SET=1
C
1     IF (I.EQ.COL) GO TO 170
      SET=0
      COL=I
C
C       FIND THE DESIRED GPA
C
170   CALL CALC(GPA,HOURS,ERROR,NUMB,1,SET)
C
C       FORGET ABOUT THIS STUDENT IF HE HASN'T TAKEN ANY COURSE
C       OF INTEREST IN THIS COMPARISON
C
      IF (NUMB.EQ.0) GO TO 3000
      GR=GPA*HOURS
      G(J)=G(J)+GR
      H(J)=H(J)+HOURS
      N(J)=N(J)+1
C
C       SAVE THIS STUDENT'S GRADES*HOURS FOR USE IN FINDING THE STANDARD
C       DEVIATION AFTER THE MEAN IS KNOWN
C
      CALL OPEN(J,FILE(J),'MCD')
      WRITE(J,200) GR
200   FORMAT(F10.3)
      CALL CLOSE(J)
C
C       GO GET THE NEXT STUDENT FILE
C
      GO TO 3000
C
C       FIND THE AVERAGE COMPOSITE GPA FOR EACH CLASS
C
1000  IF (N(1).NE.0) G(1)=G(1)/N(1)
      IF (N(2).NE.0) G(2)=G(2)/N(2)
      DO 1500 I=1,2
      CALL OPEN(I,FILE(I),'INPUT',K)
C
C       FIND THE STANDARD DEVIATIONS FOR EACH CLASS
C
C       SEE IF FILE(I) EXISTED
C
      IF (N(I).EQ.0) GO TO 1500
      K=N(I)
      DO 1400 J=1,K
      READ(I,200,END=1500) GR
```

```
      SDEV(I)=SDEV(I)+(G(I)-GR)**2
1400  CONTINUE
1500  CALL CLOSE(1)
      DO 2000 I=1,2
      SDEV(I)=SQRT(SDEV(I))
2000  WRITE(6,2100) N(I),VERB(I),DEPT2,NUM,G(I),SDEV(I)
2100  FORMAT(1X,I4,' STUDENTS ',A8,A5,I3/' GRADES*HOURS(AVE)=',%
        F8.3,5X,'STANDARD DEVIATION=',F8.3)
"
"       RE-INITIALIZE THE PROCESSOR FOR FURTHER USE
"
      START=0
      DO 2140 I=1,2
      N(I)=0
      G(I)=0.
      SDEV(I)=0.
      H(I)=0.
2140  CONTINUE
2150  CALL CLOSE(1,2,12)
      IF (ERROR.EQ.10) ERROR=1
3000  IF (DEBUG.EQ.1) WRITE(6,3200) ERROR
3200  FORMAT(' *LEAVING USERA--ERROR=',I1)
      RETURN
      END
"
"
"
"
      SUBROUTINE DUD
      RETURN
      END
"
"
"
"
"       OLF SUBPROGRAMS
"
"       SUBROUTINE LOAD LOADS FILE NAMED NAME INTO DIRECTORY SELECTED
"       BY LEVEL.  IF NAME=LAST(LEVEL), PROCESSING IS SKIPPED
"       IF COPY=1, FILE NAMED LAST(LEVEL) IS COPIED TO DISC BEFORE FILE
"       NAMED NAME IS LOADED INTO CORE
"
      SUBROUTINE LOAD(LEVEL,NAME,COPY,LAST,VECTOR,LENGTH,NAME2,KARD)
      REAL*8 NAME,VECTOR(260),LAST(4),NAME2
      INTEGER COPY,L1(3)/1,201,241/,L3(3)/0,20,0/,LENGTH(3),DEBUG/0/
      IF (DEBUG.EQ.1) WRITE(6,1) LEVEL,NAME
1     FORMAT(' *ENTERING LOAD--LEVEL=',I1,%
        ', TO FIND ',A8)
      IF (NAME.EQ.LAST(LEVEL)) GO TO 100
      IF (KARD.EQ.1) WRITE(6,4) LEVEL,NAME2,NAME
4     FORMAT(' NOW LOADING LEVEL-',I1,' DIRECTORY ',A8,' (',A8,')')
      J=L1(LEVEL)
      IF (COPY.EQ.0) GO TO 50
      IF (DEBUG.EQ.1) WRITE(6,8) LAST(LEVEL)
8     FORMAT(' MUST COPY FILE ',A8,' FIRST')
      CALL OPEN(1,LAST(LEVEL),'OUTPUT')
      K=J+L3(LEVEL)+LENGTH(LEVEL)-1
```

```
      WRITE(1,10)LENGTH(LEVEL),(VECTOR(I),I=J,K)
10    FORMAT(A4/(10A8))
      CALL CLOSE(1)
      COPY=0
50    CALL OPEN(1,NAME,'INPUT')
      READ(1,55) M
55    FORMAT(A4)
      LENGTH(LEVEL)=M
      K=J+M-1+L3(LEVEL)
      IF (M.EQ.0) K=J+L3(LEVEL)
      READ(1,60)(VECTOR(I),I=J,K)
60    FORMAT(10A8)
      LAST(LEVEL)=NAME
100   IF (DEBUG.EQ.1) WRITE(6,105)
105   FORMAT(' *LEAVING LOAD')
      CALL CLOSE(1)
      RETURN
      END
C
C
C
C
C       SUBROUTINE OUTPUT PRINTS STATUS OF SYSTEM JUST BEFORE
C       INVOKING (OR SKIPPING) THE CURRENT PROCESSOR
C
      SUBROUTINE OUTPUT(I,PROC,BATCH,DUPLIC,LAST,J,GIB8,%
        LINE,HEADER,ISTART,LEN,NAME2,OK)
      REAL*8 GIB8(5),XXX,YYY,PROC,DUPLIC(4),LAST(4),NAME2,%
        KIND(3)/'CRRCLM','ADVISOR','STUDENT'/
      INTEGER*2 HEADER(32),A(4),B(4)/4*' '/,LINE(72),%
        QUOTE/'"'/
      INTEGER BATCH,DEBUG/0/,OK
      EQUIVALENCE (YYY,A(1)),(XXX,B(1))
      OK=0
      IF (DEBUG.EQ.1) WRITE(6,1) I
1     FORMAT(' *ENTERING OUTPUT--ERROR=',I2)
      K=I+1
      IF (K.GT.2) WRITE(6,5)
5     FORMAT('0SKIPPING PROCESSING STEP')
      GO TO (10,10,30,50,60,67,70,80,80,90,67,100,110),K
10    WRITE(6,15) PROC
15    FORMAT('0ABOUT TO ENTER PROCESSOR ',A8)
      IF (K.EQ.1) WRITE(6,17) HEADER,DUPLIC(3)
17    FORMAT(' STUDENT: ',32A1,'   (',A8,')')
      IF (K.EQ.2) WRITE(6,20)
20    FORMAT(1X,'STATISTICS WRAPUP')
      OK=1
      GO TO 1000
30    WRITE(6,40)
40    FORMAT('+',24X,'--TOO MUCH INPUT')
      OK=2
      GO TO 1000
50    YYY=GIB8(2)
      B(1)=A(2)
      B(2)=A(3)
      B(3)=A(4)
      WRITE(6,55)(HEADER(K),K=10,29),XXX,GIB8(3)
```

```
55    FORMAT(' FILE BELONGS TO ',20A1,' CRRCLM=',A8,' ADVISOR=',A8)
      GO TO 1000
60    WRITE(6,65) (HEADER(M),M=1,9),GIB8(3)
65    FORMAT(1X,9A1,' WAS ACCESSED ONLY BECAUSE ',A8,' ADVISES ',%
        'SEVERAL CURRICULA.')
      GO TO 1000
67    IEND=ISTART+LEN-1
      IF (K.EQ.6) WRITE(6,68)(LINE(L),L=ISTART,IEND),QUOTE
68    FORMAT('+',26X,'CANNCT TRANSLATE "',21A1)
      OK=2
      GO TO 1000
70    K=BATCH+1
      IF (K.EQ.0) K=1
      WRITE(6,75) KIND(K)
75    FORMAT(' IMPROPER COMMAND--SPECIFY AT OR BELOW ',A7,' LEVEL')
      OK=2
      GO TO 1000
80    K=J
      IF (K.EQ.2) K=4
      WRITE(6,85) DUPLIC(J),J,LAST(J)
85    FORMAT(' FILE ',A8,' IS NOT IN LEVEL-',I1,' DIRECTORY ',A8)
      GO TO 1000
90    WRITE(6,95)(LINE(K),K=21,29),NAME2
95    FORMAT(' FILE ',9A1,' IS NOT IN LEVEL-3 DIRECTORY ',A8)
      GO TO 1000
100   WRITE(6,105)
105   FORMAT(' YOU MUST SPECIFY A NEW PROCESSOR')
      OK=2
      GO TO 1000
110   KK=J+1
      WRITE(6,115) KK,NAME2
115   FORMAT(' LEVEL-',I1,' DIRECTORY ',A8,' IS EMPTY')
1000  IF (DEBUG.EQ.1) WRITE(6,1010)
1010  FORMAT(' *LEAVING OUTPUT')
      RETURN
      END
"
"
"
"
"      FUNCTION COMP:
"      RETURNS COMP= 1 IF KEY.GT.XMBER
"      RETURNS COMP= 0 IF KEY.EQ.XMBER
"      RETURNS COMP=-1 IF KEY.LT.XMBER
"
      INTEGER FUNCTION COMP(PASS,LEVEL,XMBER,KEY)
      INTEGER PASS,DEBUG/0/,TOP(2)/18,1/,BOT(2)/44,17/,%
        MATRIX(4),BEGIN(4)/4,8,12,16/
      INTEGER*2 DIGIT(16),NUM(44)/'$','0','1','2','3',%
        '4','5','6','7','8','9','A','B','C','D','E','F',%
        ' ','A','B','C','D','E','F','G','H','I','J','K',%
        'L','M','N','O','P','Q','R','S','T','U','V','W','X',%
        'Y','Z'/
      REAL*8 KEY,XMBER,X,Y
      EQUIVALENCE(X,MATRIX(1)),(Y,MATRIX(3))
      IF (DEBUG.EQ.1) WRITE(6,5) XMBER,KEY
5     FORMAT(' *ENTERING COMP--XMBER=',A8,%
```

```
     *    KEY=',A8)
      COMP=0
      K=3
      IF (PASS.EQ.1) GO TO 20
      PASS=1
      K=1
      KK=MOD(LEVEL,2)+1
      ISTART=TOP(KK)
      IEND=BOT(KK)
      X=KEY
20    Y=XMBER
      DO 60 I=K,4
      KKK=BEGIN(I)
      MULT=0
      IF (MATRIX(I).LT.0) MULT=-1
      DIGIT(KKK)=MOD(MATRIX(I),256)*256+64
      MATRIX(I)=MATRIX(I)/256+255*MULT*16777216+MULT
      KKK=KKK-1
      DO 60 J=1,3
      DIGIT(KKK)=MOD(MATRIX(I),256)*256+64
      MATRIX(I)=MATRIX(I)/256
60    KKK=KKK-1
      DO 100 I=1,8
      DO 70 J=ISTART,IEND
      IF (DIGIT(I).EQ.NUM(J)) GO TO 75
70    CONTINUE
75    JJJ=I+8
      DO 80 JJ=ISTART,IEND
      IF (DIGIT(JJJ).EQ.NUM(JJ)) GO TO 85
80    CONTINUE
85    L=J-JJ
      IF (L.EQ.0) GO TO 100
      COMP=IABS(L)/L
      GO TO 150
100   CONTINUE
150   IF (DEBUG.EQ.1) WRITE(6,170) COMP
170   FORMAT(' *LEAVING COMP--COMP=',I2)
      RETURN
      END
"
"
"
"
"      SUBROUTINE RENAME RIGHT-SHIFTS XXX BY TWO BYTES AND
"      INSERTS CHAR INTO THE RESULTING VACANCY
"
      IF (DEBUG.EQ.1) WRITE(6,1) XXX,CHAR
1     FORMAT(' *ENTERING RENAME--XXX=',A8,', CHAR=',A2)
      XXXX=XXX
      MULT1=0
      MULT2=0
      IF (MATRIX(1).LE.0) MULT1=-1
      IF (MATRIX(2).LE.0) MULT2=-1
      MATRIX(2)=MATRIX(2)/65536+65536*(65535*MULT2%
        +MOD(MATRIX(1),65536))+MULT2
      MATRIX(1)=MATRIX(1)/65536+65536*(65535*MULT1+CHAR)+MULT1
      XXX=XXXX
```

```
      IF (DEBUG.EQ.1) WRITE(6,10) XXX
10    FORMAT(' *LEAVING RENAME--XXX=',A8)
      RETURN
      END
"
"
"
"
"      SUBROUTINE REVISE INSERTS (DELETES) KEY1 FROM DIRECTORY(LEVEL)
"      IF INCR=+1(-1).  IF LEVEL=2, IT ALSO INSERTS (DELETES) KEY2
"
      SUBROUTINE REVISE(VECTOR,LENGTH,KEY1,KEY2,PTR,LEVEL,%
        INCR,LAST,ERR,NAME2)
      REAL*8 VECTOR(260),LAST(4),CHOICE(2)/'INSERT','DELETE'/,%
        KEY1,KEY2,BLANK/' '/,NAME2
      INTEGER LENGTH(3),DUP,PTR,ERR,L1(3)/1,201,241/,L3(3)/0,20,0/,%
        LEN(3)/200,20,20/,DEBUG/0/
      KK=L3(LEVEL)
      ITOP=L1(LEVEL)
      KM=ITOP+LENGTH(LEVEL)+L3(LEVEL)-1
      IBOT=KM-KK
      IF (DEBUG.EQ.0) GO TO 10
      K=1
      IF (INCR.EQ.-1) K=2
      WRITE(6,5) CHOICE(K),KEY1,LEVEL,LAST(LEVEL),PTR
5     FORMAT(' *ENTERING REVISE--TO ',%
        A7,A8/' IN LEVEL-',I1,%
        ' DIRECTORY ',A8,' AT POSITION ',I3)
      DO 7 I=ITOP,KM
7     IF (VECTOR(I).NE.BLANK) WRITE(6,297) I,VECTOR(I)
10    IF (INCR.EQ.-1) GO TO 100
      IF (LENGTH(LEVEL).EQ.0) GO TO 30
      IF (LENGTH(LEVEL).EQ.LEN(LEVEL)) GO TO 250
      DO 20 I=PTR,IBOT
      K=IBOT+PTR-I
      KK=K+L3(LEVEL)
      IF (LEVEL.EQ.2) VECTOR(KK+1)=VECTOR(KK)
20    VECTOR(K+1)=VECTOR(K)
30    VECTOR(PTR)=KEY1
      IF (LEVEL.EQ.2) VECTOR(PTR+L3(LEVEL))=KEY2
      GO TO 270
100   DO 150 I=PTR,IBOT
      KK=I+L3(LEVEL)
      VECTOR(KK)=VECTOR(KK+1)
150   VECTOR(I)=VECTOR(I+1)
      GO TO 270
250   ERR=1
      INCR=0
      WRITE(6,260) NAME2,LAST(LEVEL),KEY1
260   FORMAT(1X,A9,'(',A8,') IS FULL--CAN''T ADD ',A8)
270   LENGTH(LEVEL)=LENGTH(LEVEL)+INCR
      IF (LENGTH(LEVEL).NE.0) GO TO 285
      WRITE(6,280) LEVEL,NAME2,LAST(LEVEL)
280   FORMAT(' LEVEL-',I1,' DIRECTORY ',A8,' (',A8,') IS NOW EMPTY')
      KLH=L1(LEVEL)
      VECTOR(KLH)=BLANK
      VECTOR(KLH+L3(LEVEL))=BLANK
```

```
285   KM=KM+INCR
      IF (DEBUG.EQ.0) GO TO 300
      WRITE(6,290) ERR
290   FORMAT(' *LEAVING REVISE--ERROR=',I1)
      DO 295 I=ITOP,KM
295   IF (VECTOR(I).NE.BLANK) WRITE(6,297) I,VECTOR(I)
297   FORMAT(' (',I3,')',5X,A8)
300   RETURN
      END
"
"
"
"
"      SUBROUTINE BISECT DOES BINARY SEARCH FOR KEY IN DIRECTORY(LEVEL)
"      IF KEY IS FOUND, DUP=1 AND PTR=POSITION IN FILE;
"      IF KEY IS NOT FOUND, DUP=0 AND PTR=POINT OF INSERTION IN FILE.
"
      SUBROUTINE BISECT(VECTOR,LENGTH,KEY,DUP,PTR,LEVEL,LAST)
      REAL*8 VECTOR(260),KEY,LAST(4),BLANK/' '/
      INTEGER DUP,PTR,LENGTH(3),TOP(3)/0,200,240/,COMP,DEBUG/0/,&
        L3(3)/0,20,0/
      IPASS=0
      DUP=0
      ITOP=TOP(LEVEL)
      IBOT=ITOP+LENGTH(LEVEL)+1
      IF (DEBUG.EQ.0) GO TO 10
      J=ITOP+1
      K=IBOT-1+L3(LEVEL)
      WRITE(6,5) KEY,LEVEL,LAST(LEVEL)
5     FORMAT(' *ENTERING BISECT--SEARCHING FOR ',&
        A8,/' IN LEVEL-',I1,' DIRECTORY ',A8)
      DO 7 I=J,K
7     IF (VECTOR(I).NE.BLANK) WRITE(6,8) I,VECTOR(I)
8     FORMAT(' (',I3,')    ',A8)
10    IF (ITOP+1.GE.IBOT) GO TO 120
      PTR=(ITOP+IBOT)/2
      IF (COMP(IPASS,LEVEL,VECTOR(PTR),KEY)) 20,100,30
20    IBOT=PTR
      GO TO 10
30    ITOP=PTR
      GO TO 10
100   DUP=1
120   IF (DUP.EQ.0) PTR=IBOT
      IF (DEBUG.EQ.0) GO TO 300
      WRITE(6,200)
200   FORMAT(' *LEAVING BISECT--')
      IF (DUP.EQ.0) WRITE(6,210) KEY,PTR
210   FORMAT('+',30X,A8,' BELONGS AT POSITION ',I3)
      IF (DUP.EQ.1) WRITE(6,220) KEY,PTR
220   FORMAT('+',30X,A8,' FOUND AT POSITION ',I3)
300   RETURN
      END
"
"
"
"
"      SUBROUTINE TIMES CONVERTS FIELDS STARTING AT P1 AND P2
```

```
"       IN LLINE TO A COMPOSITE DATE INDICATING BOTH
"       YEAR AND SEMESTER.  THE 1ST FIELD IS A SEMESTER
"       NAME AND THE 2ND FIELD IS A YEAR NUMBER.
"
      SUBROUTINE TIMES(DATE,LINE,P1,P2,FUTURE,ERROR,*)
      INTEGER P1,P2,FUTURE,ERROR,DEBUG/0/,DATE,LINE*2(72)
      REAL*8 SEASON,CHOICE(4)/'SPRING','SUMMER','FALL','AUTUMN'/,OUT
      IF (DEBUG.NE.1) GO TO 10
      DATE=0
      K1=P1+7
      K2=P2+3
      WRITE(6,5)(LINE(I),I=P2,K2),(LINE(I),I=P1,K1)
5     FORMAT(' *ENTERING TIMES',%
        ' YEAR=',4A1,5X,'SEASON=',8A1)
10    CALL CONVRT(SEASON,5,LINE,8,JUNK,P1,ERROR,&990)
      DO 30 I=1,4
      IF (SEASON.EQ.CHOICE(I)) GO TO 50
30    CONTINUE
      WRITE(6,35) SEASON
35    FORMAT(' "',A8,'" IS NOT A VALID SEMESTER NAME')
      ERROR=1
      GO TO 1000
50    IF (I.EQ.4) I=3
      CALL CONVRT(OUT,6,LINE,4,DATE,P2,ERROR,&990)
      DATE=DATE*10+I
      FUTURE=0
      CALL OPEN(1,'DATES','INPUT')
      READ(1,60) NOW
60    FORMAT(I5)
      K=DATE-NOW
      IF (NOW.NE.DATE) FUTURE=IABS(K)/K
      GO TO 1000
990   ERROR=1
1000  IF (DEBUG.EQ.1) WRITE(6,1005) DATE,FUTURE
1005  FORMAT(' LEAVING TIMES--YEAR=',I5,' FUTURE=',I2)
      CALL CLOSE(1)
      IF (ERROR.NE.0) RETURN 1
      RETURN
      END
"
"
"
"
"       SUBROUTINE CONVRT CONVERTS LEN CHARACTERS STARTING AT LINE(START)
"       (CONVERSION  TYPE DETERMINED BY VALUE OF ITYPE):
"       1) TO NAME OF STUDENT FILE (SSN IN HEXADECIMAL)
"       2) TO BASIS FOR CURRICULUM FILES (DECIMAL)
"       3) JUST CHECKS FOR INVALID CHARACTERS
"       4) TO DEPARTMENT NAME (IN OUT) & COURSE # (IN AUX)
"       5) TO BASIS FOR DEPARTMENT FILE NAMES
"          OR TO AN ADVISOR'S NAME (ALPHABETIC)
"       6) CONVERTS CHARACTER TO DECIMAL NUMBER (IN AUX)
"       7) CONVERTS # IN AUX TO HEXADECIMAL FILE
"          NAME IN OUT
"
      SUBROUTINE CONVRT(OUT,ITYPE,LINE,LEN,AUX,START,ERR,*)
      INTEGER*2 IN(20),LINE(72),NUMBER(11)/'0','1','2','3',%
```

```
     '4','5','6','7','8','9',' '/,ALPHA(30)/'A',%
     'B','C','D','E','F','G','H','I','J','K','L','M',%
     'N','O','P','Q','R','S','T','U','V','W','X','Y',%
     'Z',' ',',','''','-'/,ITEMP(2),LT,BLANK,DOLLAR/'$'/,%
     QUOTE/'"'/
     INTEGER MATRIX(2),HEX,AUX,START,DEBUG/0/,ERR
     REAL*8 OUT,FLAG/'????????'/,XXX,CLEAR(5),BL/' '/
     EQUIVALENCE (MATRIX(1),XXX),(ITEMP(1),IT),%
       (BLANK,ALPHA(27)),(IN(1),CLEAR(1))
     M=0
     ERR=0
     LLEN=LEN
     XXX=0.0D0
     HEX=268435456
     IF (LLEN.GT.20) LLEN=20
     DO 2 I=1,5
2    CLEAR(I)=BL
     K=START
     DO 5 I=1,LLEN
     IN(I)=LINE(K)
5    K=K+1
     IF (DEBUG.EQ.1) WRITE(6,6) ITYPE,IN
6    FORMAT(' *ENTERING CONVRT(ITYPE=',I1,')--',%
       'CONVERTING ',20A1)
     GO TO (10,500,1000,1700,500,10,60),ITYPE
10   NUM=0
     DO 50 I=1,LLEN
     LT=IN(I)
     DO 30 J=1,10
     IF (LT.EQ.NUMBER(J)) GO TO 40
30   CONTINUE
     GO TO 2500
40   NUM=NUM*10+J-1
50   CONTINUE
60   IF (ITYPE.EQ.7) NUM=AUX
     AUX=NUM
     IF (ITYPE.EQ.7) GO TO 70
     IF (ITYPE.NE.1) GO TO 3000
70   DO 100 I=1,2
     DO 100 J=1,4
     M=8-2*J
     L=NUM/HEX
     NUM=NUM-L*HEX
     N=16**M
     IF (L.LE.9) MATRIX(I)=MATRIX(I)+N*(L+15*16)
     IF (L.GT.9) MATRIX(I)=MATRIX(I)+N*((L-9)+12*16)
100  HEX=HEX/16
     GO TO 3000
500  L=1
     DO 600 I=1,8
     LT=IN(I)
     IT=0
     ITEMP(2)=LT
     IF (ITYPE.NE.2) GO TO 527
     DO 525 J=1,11
     IF (LT.EQ.NUMBER(J)) GO TO 540
525  CONTINUE
```

```
      GO TO 2500
527   DO 530 J=1,27
      IF (LT.EQ.ALPHA(J)) GO TO 540
530   CONTINUE
      GO TO 2500
540   IF (I.EQ.5) L=2
      IF (I.EQ.5) HEX=268435456
      MATRIX(L)=MATRIX(L)+(IT-64)/256*(HEX/16)
      HEX=HEX/256
600   CONTINUE
      IF (ITYPE.EQ.4) GO TC 1850
      GO TO 3000
1000  DO 1200 I=1,20
      LT=IN(I)
      DO 1100 J=1,30
      IF (LT.EQ.ALPHA(J)) GO TO 1200
1100  CONTINUE
      GO TO 2500
1200  CONTINUE
      GO TO 3000
1700  DO 1800 IPT=1,6
      DO 1750 J=1,26
      IF (IN(IPT).EQ.ALPHA(J)) GO TO 1800
1750  CONTINUE
      GO TO 1820
1800  CONTINUE
      GO TO 2500
1820  LLEN=IPT-1
      K=9
      DO 1830 I=IPT,8
      IN(K)=IN(I)
      IN(I)=BLANK
1830  K=K+1
      GO TO 500
1850  L=IPT+2
      K=9
      DO 1910 I=7,9
      IN(I)=IN(K)
1910  K=K+1
      NONO=NUMBER(1)
      DO 1915 I=1,6
1915  IN(I)=NCNO
      LLEN=9
      GO TO 10
2500  XXX=FLAG
      ERR=10
      WRITE(6,2600)(IN(I),I=1,LLEN),QUOTE
2600  FORMAT(' CANNOT TRANSLATE "',21A1)
3000  OUT=XXX
      IF (DEBUG.EQ.1) WRITE(6,3005) OUT,AUX
3005  FORMAT(' *LEAVING CONVRT--OUT=',A8,%
        '      AUX=',I10)
      IF (ERR.NE.0) RETURN 1
      RETURN
      END
"
"
```

```
C
C
C       OLF UTILITY PROGRAMS
C
C       DREDIT CREATES AND DELETES DIRECTORY FILES
C       SUBPROGRAMS INVOKED IN CREDIT ARE SHOWN IN THE OLF LISTING
C
      REAL*8 CURRIC/'CURRIC'/,LAST(4)/4*'DUMMY'/,X,Y,Z,%
        VECTOR(260),NAME,ENTRY(2),VERB,BLANK/' '/,ACTION(3)%
        /'INSERT','DELETE','NEXT'/,LIST(2)/'CRLIST','DLLIST'/,%
        BUF(21)/21*' '/,DUMMY/'DUMMY'/,CHOICE(2)/'FOUND','IS NOT'/,%
        THING
      REAL*4 V1(2)/'INTO','FROM'/,ANS(3)/'1','2','DONE'/
      INTEGER ERROR,PTR,DUP,TYPE(3)/2,5,1/,ACT(2)/1,-1/,%
        AUXOUT,LENGTH(3),COPY(2)/2*0/,LEN(3)/8,8,10/,%
        ZERO/0/,IRUN/0/,KIND(2)/5,2/,V(2)/' ','DEST'/,%
        LLEN(2)/21,2/
      INTEGER*2 IN(20),LINE(72),$$/'$$'/,A(4),ONE/'1'/,%
        BL/' '/
      COMPLEX*16 WORDS(2)/'CURRICULUM #','NAME AND SSN'/,%
        BUFF,CLEAR/' '/
      EQUIVALENCE (IN(1),LINE(1)),(NAME,A(1)),%
        (BUF(1),BUFF)
C
C       LOAD LEVEL-1 DIRECTORY CURRIC
C
      CALL LOAD(1,CURRIC,0,LAST,VECTOR,LENGTH,CURRIC,0)
5     M=1
C
C       REQUEST THE NAME OF THE TO-BE-EDITED FILE
C
      WRITE(6,10)
10    FORMAT(' TYPE LEVEL--1 OR 2 (OR DONE) AND NAME OF FILE',%
        ' TO  BE EDITED'/)
      READ(5,15) ANSWER,IN
15    FORMAT(A4,6X,20A1)
      DO 20 LEVEL=1,3
      IF (ANS(LEVEL).EQ.ANSWER) GO TO 40
20    CONTINUE
23    WRITE(6,25)
25    FORMAT(' SORRY--COULDN''T UNDERSTAND THAT')
27    GO TO (5,100),M
40    IF (LEVEL.EQ.3) GO TO 500
      CALL CONVRT(NAME,KIND(LEVEL),LINE,8,AUXOUT,1,ERROR,&27)
      IF (LEVEL.EQ.2) GO TO 45
      IF (NAME.NE.CURRIC) GO TO 23
      GO TO 100
C
C       PREFACE CURRICULUM CODE WITH $$
C
45    CALL RENAME($$,NAME)
C
C       IS THE REQUESTED FILE KNOWN?
C
      CALL BISECT(VECTOR,LENGTH,NAME,DUP,PTR,1,LAST)
      IF (DUP.EQ.1) GO TO 90
      WRITE(6,50) A(2),A(3),A(4)
```

```
50    FORMAT(1X,3A2,' IS AN UNKNOWN CURRICULUM')
      GO TO 5
C
C      LOAD THE TO-BE-EDITED FILE
C
90    CALL LOAD(2,NAME,COPY(2),LAST,VECTOR,LENGTH,NAME,0)
100   M=2
C
C      DETERMINE THE TYPE OF EDITING REQUIRED
C      DETERMINE THE NAME OF THE FILE TO BE INSERTED (DELETED)
C
      WRITE(6,110) WORDS(LEVEL)
110   FORMAT(' TYPE ACTION, ',A12)
      READ(5,120) VERB,(IN(I),I=1,8),(IN(I),I=12,20)
120   FORMAT(A8,2X,8A1,2X,5A1)
      DO 130 I=1,3
      IF (VERB.EQ.ACTION(I)) GO TO 150
130   CONTINUE
      GO TO 23
C
C      IF TYPE OF EDITING = NEXT, GO GET THE NAME OF THE
C      NEXT FILE TO BE EDITED
C
150   IF (I.EQ.3) GO TO 5
      INCR=ACT(I)
      J=0+LEVEL
      K=1
160   CALL CONVRT(ENTRY(K),TYPE(J),LINE,LEN(J),AUXOUT,%
        (K-1)*10+1,ERROR,&27)
      J=3
      IF (LEVEL.NE.2) GO TO 200
      IF (K.EQ.2) IN(11)=BL
      IF (K.EQ.2) GO TO 200
      K=2
C
C      ADD 1 BILLION TO THE ADVISOR'S SSN TO MAKE ADVISOR FILENAME
C      CLEARLY DISTINGUISHABLE FROM STUDENT FILENAME
C
      IN(11)=ONE
      GO TO 160
200   IF (LEVEL.EQ.1) CALL RENAME($$,ENTRY(1))
C
C      DOES THE FILE TO BE INSERTED (DELETED) EXIST?
C
      CALL BISECT(VECTOR,LENGTH,ENTRY(1),DUP,PTR,LEVEL,LAST)
      IF (I-DUP.EQ.1) GO TO 220
C
C      FILE EITHER EXISTS WHEN IT SHOULDN'T OR
C      DOESN'T EXIST WHEN IT SHOULD
C
      WRITE(6,205) ENTRY(1),CHOICE(I),LAST(LEVEL),VERB
205   FORMAT(1X,A8,1X,A6,' IN DIRECTORY ',A8,%
        ' CAN''T ',A6,' IT')
      GO TO 100
220   IF (LEVEL.EQ.1) GO TO 245
      IF (INCR.EQ.-1) GO TO 245
C
```

```
C       CREATING A NEW FILE--MAKE SURE THAT THE PROPOSED
C       FILE DOES NOT EXIST ON ANOTHER BRANCH OF THE OLF TREE
C
      CALL OPEN(1,ENTRY(2),'INPUT',N)
      IF (N.NE.0) GO TO 243
C
C       THE FILE EXISTS
C
      READ(1,270) N
      IF (N.EQ.0) GO TO 235
C
C       FIND THE NAME AND CURRICULUM OF THE PRESENT
C       OWNER OF THE DISPUTED SSN FROM A STUDENT FILE
C
      READ(1,275) X
      CALL OPEN(2,X,'INPUT')
      READ(2,230) Y,Z
230   FORMAT(40X,2A8)
      CALL CLOSE(2)
      GO TO 240
C
C       IF THIS IS A NEW ADVISOR FILE, IT HAS
C       A HEADER SHOWING OWNER'S NAME AND CURRICULUM
C
235   READ(1,275) Z,Y
240   WRITE(6,242)(IN(N),N=12,20),Z,Y
242   FORMAT(1X,9A1,' IS THE SSN LISTED FOR ',A8,' OF ',$
        'CURRICULUM ',A8)
      CALL CLOSE(1)
      GO TO 100
243   CALL CLOSE(1)
C
C       INSERT (DELETE) THE FILE
C
245   CALL REVISE(VECTOR,LENGTH,ENTRY(1),ENTRY(2),PTR,LEVEL,$
        INCR,LAST,ERROR,ENTRY(K))
      IF (ERROR.EQ.1) GO TO 100
      IF (INCR.EQ.0) GO TO 300
C
C       CREATE THE NEW FILE
C
      CALL OPEN(1,ENTRY(K),'OUTPUT')
      WRITE(1,270) ZERO
270   FORMAT(A4)
      IF (LEVEL.EQ.1) GO TO 273
      BUF(1)=ENTRY(1)
      BUF(2)=NAME
273   KK=LLEN(LEVEL)
      WRITE(1,275)(BUF(J),J=1,KK)
275   FORMAT(10A8)
      IF (LEVEL.EQ.2) BUFF=CLEAR
      CALL CLOSE(1)
C
C       ADD FILE TO A BACKUP LIST (CRLIST OR DLLIST)
C
      CALL OPEN(1,LIST(I),'MOD')
      THING=BLANK
```

```
      IF (LEVEL.EQ.2) THING=ENTRY(1)
      WRITE(1,280) V(I),ENTRY(K),THING,V1(I),NAME
280   FORMAT(A4,1X,A8,1X,A8,1X,A4,1X,A8)
      CALL CLOSE(1)
C
C      ENABLE DIRECTORY REWRITE
C
300   COPY(LEVEL)=1
      IRUN=1
C
C      GET NEXT COMMAND
C
      GO TO 100
500   DO 550 I=1,2
C
C      MAKE SURE ALL DIRECTORIES HAVE BEEN REWRITTEN BEFORE STOPPING
C
550   IF (COPY(I).EQ.1) CALL LOAD(I,DUMMY,COPY(I),LAST,%
        VECTOR,LENGTH,DUMMY,0)
C
C      IF ANY FILES HAVE BEEN EDITED, REQUEST A TAPE COPY
C
      IF (IRUN.EQ.1) WRITE(6,560)
560   FORMAT(' AFTER JOB ENDS--TYPE:  RUNN COPY')
      STOP
      END
C
C
C
C
C      RECOVR EDITS DIRECTORIES AFTER SYSTEM FAILURE TO
C      CONFORM TO FILES CRLIST AND DLLIST
C      SUBPROGRAMS INVOKED BY RECOVR ARE SHOWN IN THE OLF LISTING
C
      INTEGER LENGTH(3),ACT(2)/1,-1/,DUP,PTR,ERR,%
        COPY(3)/3*0/,OK(2)/0,1/
      INTEGER*2 $$/'$$'/,A
      REAL*8 VECTOR(260),LAST(4)/4*'DUMMY'/,DUMMY/'DUMMY'/,BLANK/' '/,%
        LIST(2)/'CRLIST' ,'DLLIST'/,ITEM1,ITEM2,CURRIC/'CURRIC'/,FILE
      EQUIVALENCE (A,FILE)
      DO 1000 I=1,2
      INCR=ACT(I)
      IOK=OK(I)
      CALL OPEN(10,LIST(I),'INPUT')
      DO 500 J=1,10000000
      READ(10,10,END=1000) ITEM2,ITEM1,FILE
10    FORMAT(5X,A8,1X,A8,6X,A8)
      LEVEL=3
      IF (FILE.EQ.CURRIC) LEVEL=1
      IF (A.EQ.$$) LEVEL=2
      CALL LOAD(LEVEL,FILE,COPY(LEVEL),LAST,VECTOR,LENGTH,BLANK,0)
      IF (ITEM1.EQ.BLANK) ITEM1=ITEM2
      COPY(LEVEL)=1
      CALL BISECT(VECTOR,LENGTH,ITEM1,DUP,PTR,LEVEL,LAST)
      IF (IOK.EQ.DUP) CALL REVISE(VECTOR,LENGTH,%
        ITEM1,ITEM2,PTR,LEVEL,INCR,LAST,ERR,FILE)
500   CONTINUE
```

```
1000  CALL CLOSE(10)
      DO 1050 I=1,3
1050  IF (COPY(LEVEL).EQ.1) CALL LOAD(I,DUMMY,1,LAST,VECTOR,%
        LENGTH,DUMMY,0)
      STOP
      END



       COPY IS A BATCH JOB WHICH INVOKES SEVERAL
       PROGRAMS TO CREATE TAPE COPIES OF OLF FILES
       FOR BACKUP PURPOSES

/** ID  CARD IMAGES GO HERE
// EXEC TSBATCH
//SYSIN DD *
(PLORTS LOGIN INFORMATION GOES HERE)
OPEN CCOPY
FORTRAN
CLOSE
RUNN ID,COPY1,SAVE,SLSTAR
RUNN ID,COPY2,SAVE,SLSTAR,DELETE1,CLLIST,DELETE2
LOGOUT
/*
"
"       CCOPY CREATES A FILE (SAVE) WHICH IS A COMPOSITE
"       OF ALL OLF DATA FILES
"
      REAL*8 VECTOR(260),GIB8(5),DCODE(12),NAME(4)/'CURRIC',3*' '/
      REAL GIB4(16)
      INTEGER*2 GIB2(20),CHEKOV(12),HEADER(32),DBLOCK(56,12)
      INTEGER PART(5,5),LENGTH(3),L1(3)/1,201,241/,L3(3)/0,20,0/,%
        LEN(3),ROWS/5/
      CALL OPEN(10,'SAVE','OUTPUT')
      CALL OPEN(1,'DATES','INPUT')
      READ(1,5) I,J
5     FORMAT(2I5)
      WRITE(10,5) I,J
      CALL OPEN(2,'PART','INPUT')
      READ(2,10)((PART(I,J),J=1,5),I=1,ROWS)
10    FORMAT(5(I1,1X))
      WRITE(10,10)((PART(I,J),J=1,5),I=1,ROWS)
      CALL CLOSE(1,2)
      I=1
12    LEN(I)=0
      CALL OPEN(I,NAME(I),'INPUT')
      READ(I,15) K
15    FORMAT(A4)
      WRITE(10,15) K
      LENGTH(I)=K
      IF (LENGTH(1).EQ.0) GO TO 80
      IF (K.EQ.0) GO TO 60
17    LEN(I)=LEN(I)+1
      IF (LEN(I).NE.1) GO TO 22
      ISTART=L1(I)
      IEND=L1(I)+L3(I)+K-1
```

```
      IF (LEVEL.EQ.2) THING=ENTRY(I)
      WRITE(1,280) V(I),ENTRY(K),THING,V1(I),NAME
280   FORMAT(A4,1X,A8,1X,A8,1X,A4,1X,A8)
      CALL CLOSE(1)
C
C      ENABLE DIRECTORY REWRITE
C
300   COPY(LEVEL)=1
      IRUN=1
C
C      GET NEXT COMMAND
C
      GO TO 100
500   DO 550 I=1,2
C
C      MAKE SURE ALL DIRECTORIES HAVE BEEN REWRITTEN BEFORE STOPPING
C
550   IF (COPY(I).EQ.1) CALL LOAD(I,DUMMY,COPY(I),LAST,%
        VECTOR,LENGTH,DUMMY,0)
C
C      IF ANY FILES HAVE BEEN EDITED, REQUEST A TAPE COPY
C
      IF (IRUN.EQ.1) WRITE(6,560)
560   FORMAT(' AFTER JOB ENDS--TYPE:  RUNN COPY')
      STOP
      END
C
C
C
C
C      RECOVR EDITS DIRECTORIES AFTER SYSTEM FAILURE TO
C      CONFORM TO FILES CRLIST AND DLLIST
C      SUBPROGRAMS INVOKED BY RECOVR ARE SHOWN IN THE OLF LISTING
C
      INTEGER LENGTH(3),ACT(2)/1,-1/,DUP,PTR,ERR,%
        COPY(3)/3*0/,OK(2)/0,1/
      INTEGER*2 $$/'$$'/,A
      REAL*8 VECTOR(260),LAST(4)/4*'DUMMY'/,DUMMY/'DUMMY'/,BLANK/' '/,%
        LIST(2)/'CRLIST' ,'DLLIST'/,ITEM1,ITEM2,CURRIC/'CURRIC'/,FILE
      EQUIVALENCE (A,FILE)
      DO 1000 I=1,2
      INCR=ACT(I)
      IOK=OK(I)
      CALL OPEN(10,LIST(I),'INPUT')
      DO 500 J=1,10000000
      READ(10,10,END=1000) ITEM2,ITEM1,FILE
10    FORMAT(5X,A8,1X,A8,6X,A8)
      LEVEL=3
      IF (FILE.EQ.CURRIC) LEVEL=1
      IF (A.EQ.$$) LEVEL=2
      CALL LOAD(LEVEL,FILE,COPY(LEVEL),LAST,VECTOR,LENGTH,BLANK,0)
      IF (ITEM1.EQ.BLANK) ITEM1=ITEM2
      COPY(LEVEL)=1
      CALL BISECT(VECTOR,LENGTH,ITEM1,DUP,PTR,LEVEL,LAST)
      IF (IOK.EQ.DUP) CALL REVISE(VECTOR,LENGTH,%
        ITEM1,ITEM2,PTR,LEVEL,INCR,LAST,ERR,FILE)
500   CONTINUE
```

```
      READ(1,20)(VECTOR(J),J=ISTART,IEND)
20    FORMAT(10A8)
      WRITE(10,20)(VECTOR(J),J=ISTART,IEND)
22    NAME(I+1)=VECTOR(L1(I)+L3(I)+LEN(I)-1)
      I=I+1
      IF (I.LE.3) GO TO 12
      CALL CLOSE(1)
      DO 40 J=ISTART,IEND
      CALL OPEN(1,VECTOR(J),'INPUT')
      READ(1,25) HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK
25    FORMAT(32A1,5A8/16A4/20A2,12A1/6A8/6A8/(28A2))
40    WRITE(10,25) HEADER,GIB8,GIB4,GIB2,CHEKOV,DCODE,DBLOCK
      I=I-1
60    I=I-1
      IF (LEN(I).LT.LENGTH(I)) GO TO 17
      IF (I.GT.1) GO TO 60
80    CALL CLOSE(1,2,3,10)
      WRITE(6,100)
100   FORMAT(' ALL FILES PREPARED FOR COPY')
      STOP
      END
"
"
"
"
```

ID IS A PLORTS FILE CONTAINGING ID CARD IMAGES

COPY1 AND COPY2 ARE PLORTS FILES CONTAINING CARD IMAGES WHICH INVOKE THE CATALOGUED PROCEDURE COPY. COPY1 CREATES A COPY OF THE FILE SAVE ON THE FIRST BACKUP TAPE, COPY2 CREATES A COPY OF SAVE ON THE SECOND BACKUP TAPE. DUPLICATE COPIES PREVENT THE DESTRUCTION OF THE BACKUP SYSTEM BY A SYSTEM FAILURE DURING PREPARATION OF A NEW BACKUP TAPE

COPY1:

```
/*SETUP UNIT=TAPE,ID=027255,RING
// EXEC COPY
//SYSUT2 DD DSN=LERNER.P6416.COPY1,DISP=(NEW,KEEP),UNIT=TAPE,
//        VOL=SER=027255,LABEL=(2,SL),
//        DCB=(LRECL=80,BLKSIZE=3200,RECFM=FB)
//SYSUT1 DD *
```

COPY2:

```
/*SETUP UNIT=TAPE,ID=027310,RING
// EXEC COPY
//SYSUT2 DD DSN=LERNER.P6416.COPY2,DISP=(NEW,KEEP),UNIT=TAPE,
//        VOL=SER=027310,LABEL=(1,SL),
//        DCB=(LRECL=80,BLKSIZE=3200,RECFM=FB)
//SYSUT1 DD *
```

```
      SLSTAR IS A PLORTS FILE CONTAINING THE CARD IMAGE: /*



      DELETE1 STARTS A NEW JOB STEP TO DEALLOCATE OBSOLETE
      FILES.  AFTER THE BACKUP HAS BEEN MADE AND THE DEALLOCATION
      LIST HAS BEEN USED, SAVE, CRLIST, AND DLLIST ARE ALL OBSOLETE

// EXEC TSBATCH
//SYSIN DD *
(PLORTS LOGIN INFORMATION GOES HERE)



      DELETE2 DOES THE FINAL DEALLOCATION AND A LOGOUT

DEST CRLIST
DEST DLLIST
DEST SAVE
LOGOUT
/*



      RESTORE IS A BATCH JOB WHICH USES TWO PROGRAMS TO
      RESTORE OLF FILES TO A TIME BEFORE A SYSTEM FAILURE.
      THE BACKUP TAPE CREATED BY COPY IS USED FOR THE RESTORATION

//*  ID CARD IMAGES GO HERE
/*SETUP UNIT=TAPE,ID=027255,NORING
// EXEC FILE,PARM='6416,LERNER'
//SAVE   DD DSN=LERNER.P6416.COPY1,DISP=(SHARE,KEEP),UNIT=TAPE,
//       VOL=SER=027255,LABEL=(2,SL)
//SYSIN DD *
DD=SAVE,PRINT=YES
/*
// EXEC TSBATCH
//SYSIN DD *
(PLORTS LOGIN INFORMATION GOES HERE)
C RECOPY
FORTRAN
C
LOGOUT
/*



"     RECOPY READS A FILE (SAVE) WHICH IS A COMPOSITE OF ALL
"     OLF DATA FILES AT SOME EARLIER TIME, AND RECREATES THE
"     THE INDIVIDUAL FILES
"
      REAL*8 VECTOR(260),DCODE(12),NAME(4)/'CURRIC',3*' '/,%
```

```
        GI88(5),BLANK/' '/
      REAL GIB4(16)
      INTEGER*2 HEADER(32),CHEKOV(12),GIB2(20),DBLOCK(56,12)
      INTEGER ROWS/5/,PART(5,5),LENGTH(3),L1(3)/1,201,241/,L3(3)%
        /0,20,0/,LEN(3)
      CALL OPEN(10,'SAVE','INPUT')
      CALL OPEN(1,'DATES','OUTPUT')
      READ(10,5) I,J
5     FORMAT(2I5)
      WRITE(1,5) I,J
      CALL OPEN(2,'PART','OUTPUT')
      READ(10,10)((PART(I,J),J=1,5),I=1,ROWS)
10    FORMAT(5(I1,1X))
      WRITE(2,10)((PART(I,J),J=1,5),I=1,ROWS)
      CALL CLOSE(1,2)
      I=1
12    LEN(I)=0
      CALL OPEN(I,NAME(I),'OUTPUT')
      READ(10,15) K
15    FORMAT(A4)
      WRITE(I,15) K
      LENGTH(I)=K
      IF (K.EQ.0) WRITE(I,20) BLANK
      IF (LENGTH(1).EQ.0) GO TO 80
      IF (K.EQ.0) GO TO 60
17    LEN(I)=LEN(I)+1
      IF (LEN(I).NE.1) GO TO 22
      ISTART=L1(I)
      IEND=L1(I)+L3(I)+K-1
      READ(10,20)(VECTOR(J),J=ISTART,IEND)
20    FORMAT(10A8)
      WRITE(I,20)(VECTOR(J),J=ISTART,IEND)
22    NAME(I+1)=VECTOR(L1(I)+L3(I)+LEN(I)-1)
      I=I+1
      IF (I.LE.3) GO TO 12
      CALL CLOSE(1)
      DO 40 J=ISTART,IEND
      CALL OPEN(1,VECTOR(J),'OUTPUT')
      READ(10,25) HEADER,GI88,GIB4,GIB2,CHEKOV,DCODE,DBLOCK
25    FORMAT(32A1,5A8/16A4/20A2,12A1/6A8/6A8/(28A2))
40    WRITE(1,25) HEADER,GI88,GIB4,GIB2,CHEKOV,DCODE,DBLOCK
      I=I-1
60    I=I-1
      IF (LEN(I).LT.LENGTH(I)) GO TO 17
      IF (I.GT.1) GO TO 60
80    CALL CLOSE(1,2,3,10)
      WRITE(6,100)
100   FORMAT(' FILES ARE NOW RESTORED')
      STOP
      END
```

BIBLIOGRAPHY

[1] Data Processing for Education Administration. Detroit: American Data Processing, Inc., 1969.

[2] Knuth, Donald E. The Art of Computer Programming. Vol. I: Fundamental Algorithms. Addison-Wesley Series in Computer Science and Information Processing, ed. by Richard S. Varga and Michael A. Harrison. Reading, Massachusetts: Addison-Wesley Publishing Company, 1968.

[3] Lancaster, F. Wilfrid. Information Retrieval Systems: Characteristics, Testing, and Evaluation. Information Sciences Series, ed. by Robert M. Hayes and Joseph Becker. New York: John Wiley & Sons, Inc., 1968.

[4] PLORTS Guide: A Guide to Using the Computing Services Office Low-Cost Terminal Filing and Remote Job Entry Facility for IBM System/360. Urbana, Illinois: Computing Services Office, University of Illinois at Urbana-Champaign, 1972.

[5] Watson, Richard W. Timesharing System Design Concepts. McGraw-Hill Computer Science Series, ed. by Richard W. Hamming and Edward A. Feigenbaum. New York: McGraw-Hill Book Company, 1970.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-73-582	2.	3. Recipient's Accession No.
4. Title and Subtitle ØN-LINE FILING (ØLF) A PRØGRAM PACKAGE FØR STUDENT RECØRDS			5. Report Date October 1973 6.
7. Author(s) Edward Mark Lerner			8. Performing Organization Rept. No. UIUCDCS-R-73-582
9. Performing Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address University of Illinois at Urbana-Champaign Department of Computer Science Urbana, Illinois 61801			13. Type of Report & Period Covered Master's Thesis 14.

15. Supplementary Notes

16. Abstracts

ØLF is a set of FØRTRAN programs which automates the maintenance and use of student records. Processing of information may be done in either batch or interactive mode. ØLF is designed to facilitate the expansion of its data processing capabilities.

17. Key Words and Document Analysis. 17a. Descriptors

Data Processing
Education
Information Retrieval

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	100
	20. Security Class (This Page) UNCLASSIFIED	22. Price